Dezentral koordinierte Zugriffskontrolle in Peer-to-Peer Datenbanken

Dissertation

der Wirtschaftswissenschaftlichen
Fakultät
der Universität Zürich

zur Erlangung der Würde
eines Doktors der Informatik

vorgelegt von

Christoph Sturm
aus
Deutschland

genehmigt auf Antrag von

PROF. DR. BURKHARD STILLER
PROF. DR. MARC H. SCHOLL

2010

Die Wirtschaftswissenschaftliche Fakultät der Universität Zürich, Lehrbereich Informatik, gestattet hierdurch die Drucklegung der vorliegenden Dissertation, ohne damit zu den darin ausgesprochenen Anschauungen Stellung zu nehmen.

Zürich, den 27. Oktober 2010[1]

Der Lehrbereichsvorsteher: Prof. Dr. Harald C. Gall

[1] Datum der Promotionstermins

Abstract

This thesis introduces a distributed coordinated access control mechanism for Peer-to-Peer (P2P) databases. Since P2P systems have no central component, also the access control mechanism has to be distributed. In contrast to all known P2P access control systems, our Peer Access Control System (PACS) is created in a bottom-up manner by coordinating the local access control components of the participants. For the first time, thus, already existing local access control policies can be used to build up a global P2P access control mechanism.

To enable such coordination, all participants first export parts of their local access control policies as eXtensible Access Control Markup Language (XACML) policies. Thereby, each peer publishes information about its local users and their privileges on local objects. This information is used to grant access control privileges on local data objects to users from remote peers, the so-called global privileges. These global privileges create connections between the published XACML policies and thereby establish a coordinated global access control mechanism. Our global access control model is a combination of discretionary and role based access control with administrative delegation support.

The missing global authority implies that there may be malicious peers in the network. The management and enforcement of the global privileges has to be guaranteed even in the worst case scenario. The management of the global privileges requires the creation of a global distributed privilege store.

This thesis presents two privilege stores, an unstructured P2P privilege store and a distributed hashtable (DHT) privilege store. The P2P privilege store is based on an unstructured P2P network and uses active replication with a dedicated replication group. The replication group has the ability to manage itself and deal with network changes. The DHT privilege store, on the other hand, employs a reliable DHT. This DHT is based on a Chord network that is extended with a redundant routing algorithm and a failure test to cope with malicious peers. The reliable DHT is generated by propagating these extentions further to the DHT application.

Not only the global privilege management but also the global privilege enforcement is distributed among the peers. To enable this, a server side enforcement and a client side enforcement approach are presented. The client side enforcement approach stores and delivers only encrypted data objects and therefore, in contrast to the server side approach, supports secure data replication. In consequence, just users who possess the corresponding decryption key can access a data object. Keys are managed by a group of delegates, by using a threshold signature schema to generate and distribute the keys.

The overhead of the presented privilege stores during enforcement and privilege management using both client side or server side enforcement is experimentally evaluated with simulations. For either privilege store it is shown that they exhibit high reliability even in the worst case scenario. Besides this, the costs expressed in number of messages for storing and

querying data objects as well as for group changes are measured for different network sizes. Comparing the DHT with the unstructured P2P privilege store shows that the DHT privilege store is the more efficient solution for both enforcement approaches.

PACS is the first access control mechanism suitable for P2P systems that uses the locally available access control policies. As PACS supports extended access control models the application range of P2P databases is expanded. In addition, the client side enforcement approach designed for PACS is the first client side approach that supports extended access control models, therefore widening the application areas of this enforcement technique in P2P systems. Major experimental evaluations in this thesis provide clear evidence that sophisticated distributed access control, as provided by PACS, can be successfully established with reasonable efforts.

Zusammenfassung

Die vorliegende Arbeit thematisiert die Zugriffskontrolle für Peer-to-Peer (P2P) Systeme im allgemeinen und P2P-Datenbanken im speziellen. Diese besitzen per Definition keine zentrale Komponente und erfordern damit eine dezentral zu organisierende Zugriffskontrolle. Mit dem Peer Access Control System (PACS) wird eine neuartige Zugriffskontrolle für P2P-Datenbanken vorgestellt, die, im Gegensatz zu allen bislang bekannten Kontrollsystemen, die Zugriffskontrolle durch die Koordination der lokalen Zugriffskontrollkomponenten der einzelnen Peers etabliert. Damit ist es erstmals möglich, die schon vorhandenen lokalen Zugriffskontrollregeln auf den Peers für den Aufbau der globalen Zugriffskontrolle zu nutzen.

Die Koordination der Peers wird durch ein gemeinsames Austauschformat möglich, die eXtensible Access Control Markup Language (XACML). Jeder Peer veröffentlicht Teile seiner Zugriffskontrollregeln, d.h. Informationen zu seinen lokalen Benutzern und deren Berechtigungen an lokalen Datenobjekten. Das Erteilen globaler Privilegien an Benutzer anderer Peers ist damit möglich. Die Zugriffskontrollkomponenten der einzelnen Teilnehmer werden so zu einer koordinierten globalen Zugriffskontrolle verknüpft. Als globales Zugriffskontrollmodell setzt PACS hierbei auf eine Kombination aus Discretionary Access Control mit administrativer Delegation und rollenbasierter Zugriffskontrolle.

Aufgrund der fehlenden zentralen Instanz muss im Netzwerk mit bösartigen Knoten gerechnet werden. Die verteilte Verwaltung und Durchsetzung der globalen Privilegien muss auch beim maximalen realistischen Bedrohungsszenario sichergestellt sein. Die Verwaltung der erteilten globalen Privilegien erfordert einen verlässlichen verteilten Privilegienspeicher.

Diese Arbeit entwirft und realisiert zwei Arten von Privilegienspeichern. Der unstrukturierte P2P-Privilegienspeicher beruht auf einem unstrukturierten P2P-Netzwerk, das durch eine neuartige Form aktiver Replikation mit autonomen sich selbst verwaltenden Replikationsgruppen zu einem verlässlichen Privilegienspeicher erweitert wird.

Der DHT-Privilegienspeicher nutzt eine verlässliche verteilte Hashtabelle (DHT). Diese basiert auf der Erweiterung des zugrundeliegenden Chord Protokolls mittels Übertragung eines redundanten Routing Algorithmus und eines Fehlertests. Durch Weiterführung dieser Erweiterungen in der DHT-Anwendung entsteht eine verlässliche DHT.

Neben der Verwaltung erfolgt auch die Durchsetzung der Privilegien in PACS dezentral. Hierzu wird eine serverseitige und eine clientseitige Durchsetzung vorgestellt. Die clientseitige Durchsetzung unterstützt eine sichere Replikation der Daten, da diese verschlüsselt im Netzwerk gespeichert werden. Die Schlüsselverwaltung erfolgt durch ein schwellenkryptographisches Verfahren. Eine Gruppe von Stellvertretern sorgt für die Verteilung der Schlüssel und damit für die Durchsetzung der Privilegien.

Die vorgestellten Privilegienspeicher und der Aufwand, den diese bei der Durchsetzung und Verwaltung der Privilegien in den jeweiligen Durchsetzungsvarianten erzeugen, wird anhand von Simulationen experimentell evaluiert. Hierbei wird die Verlässlichkeit beider Speicherty-

pen im zuvor bestimmten Bedrohungsrahmen nachgewiesen. Darüber hinaus wird der Aufwand zum Speichern und Abrufen von Datenobjekten sowie bei Gruppenänderungen in verschiedenen Netzwerkgrößen gemessen. Sowohl bei serverseitiger als auch clientseitiger Durchsetzung arbeitet der DHT-Privilegienspeicher effizienter als der P2P-Privilegienspeicher.

Durch die Nutzung lokaler Zugriffskontrollregeln stellt PACS gegenüber den momentanen Zugriffskontrollsystemen von P2P-Systemen einen wesentlichen Fortschritt dar. Da PACS zudem erweiterte Zugriffskontrollmodelle unterstützt, wird das Anwendungsgebiet von P2P-Datenbanken vergrößert. Die clientseitige Durchsetzung von PACS ist die erste clientseitige Durchsetzungsvariante, die auch erweiterte Zugriffskontrollmodelle unterstützt. Hierdurch erweitert PACS den Anwendungsbereich dieser Durchsetzungsvariante und der P2P-Systeme, die diese einsetzen. Die experimentellen Evaluationen belegen nachdrücklich, dass eine dezentral koordinierte Zugriffskontrolle mit Unterstützung erweiterter Zugriffskontrollmodelle für P2P-Systeme mit vertretbarem Aufwand realisiert werden kann.

Danksagung

Diese Arbeit entstand während meiner Anstellung als wissentschaftlicher Mitarbeiter der Datenbankgruppe des Instituts für Informatik der Universität Zürich.

Ich möchte mich deshalb bei Prof. Klaus R. Dittrich bedanken, der mir die Möglichkeit zur Realisierung meiner Forschung eröffnete und diese bis zu seinem plötzlichen Tod betreute. Er war es, der mich in seiner unnachahmlich überzeugenden Art für die Forschung im Bereich Datensicherheit und P2P-Datenbanken begeisterte und so das Fundament für das Entstehen dieser Arbeit legte.

Daneben gebührt Prof. Carl-Christian Kanne mein Dank, der die Datenbankgruppe kommissarisch leitete, sowie Prof. Michael Böhlen, der die Gruppe anschließend übernahm. Alle Professoren des Datenbanklehrstuhls haben mich immer auf meinem Weg unterstützt und mir so die nötige Sicherheit für meine Forschung gegeben. Auch danke ich dem Direktor des Instituts für Informatik, Prof. Martin Glinz, der entscheidend dazu beigetragen hat, dass die Datenbankgruppe funktionstüchtig blieb und somit die in dieser Arbeit vorgestellte Forschung fortgeführt werden konnte.

Entscheidende Bedeutung für diese Arbeit hatte die Übernahme der Betreuung meiner Forschung durch Prof. Marc H. Scholl der Universität Konstanz. Neben seinem Interesse an meiner Forschung, danke ich Ihm insbesondere für die vielen aufschlussreichen Diskussionen und wertvollen Hinweisen während dieser Zeit.

Ich danke zudem Prof. Burkhard Stiller der Universität Zürich für seine Bereitschaft, die Betreuung meiner Dissertation zu übernehmen und sein stets detailliertes und fundiertes Feedback zu dieser Arbeit.

Auch möchte ich mich für die Unterstützung meiner Kollegen der Datenbankforschungsgruppe bedanken, die jederzeit ein offenes Ohr für wissenschaftliche und technische Fragen hatten. Ebenso gebührt den Adminstratoren des IFIs und der Communication Systems Group (insbesondere Cristian Morariu) mein Dank für die Bereitstellung der Rechenerressourcen. Nicht unerwähnt bleiben soll auch Dr. Ela Hunt, die stets ein offenes Ohr für meine Probleme hatte und auch nach Jahren nicht aufgab, an meinem Englisch zu feilen.

Besonderer Dank gilt meinem ehemaligen Bürokollegen Dr. Patrick Ziegler für seine stete Hilfsbereitschaft, die unzähligen fruchtbaren wissenschaftlichen Diskussionen, seine nützlichen Hinweise und nicht zuletzt für das Korrekturlesen dieser Dissertation.

Stets eine große Unterstützung war mir meine Eltern und meine Familie. Neben ihrem Verständnis für meine notorischen Zeitknappheit während der Entstehung dieser Dissertation hat mich Ihr Vertrauen in mich immer wieder in meinem Tun bestärkt.

Bleibt mir noch, mich bei der wohl wichtigsten Person in meinem Leben zu bedanken, meiner Frau Stefanie. Ohne ihre Unterstützung, ihren unerschütterlichen Optimismus und ihr Verständnis für die stark rationierte gemeinsame Zeit, wäre diese Arbeit nicht möglich gewesen. Zum Dank hierfür widme ich Ihr diese Arbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Schon immer ist der Mensch bestrebt, seine Besitztümer vor dem Zugriff Unberechtigter zu schützen. Um dies zu gewährleisten, wird eine Fülle von Maßnahmen ergriffen, wie das Verstecken an geheimen Plätzen, der Einsatz von Schlössern, das Errichten von Mauern und Zäunen und das Aufstellen von Wachen. Nie jedoch war die Beschränkung des Zugriffs von so großer Bedeutung wie im Informationszeitalter. Da Geld- und Warenaustausch sowie Kommunikation zunehmend elektronisch stattfinden, ist eine Verlagerung der Zugriffskontrolle in Informationssysteme erforderlich. Zugriffskontrolle in Datenbanken hat dabei die Aufgabe, die Sicherheit der in den Datenbanken verwalteten Daten zu gewährleisten, so dass die potentiell sehr sensiblen Daten nur berechtigten (autorisierten) Personen zugänglich sind.

Trotz vielfältiger Gegenmaßnahmen kommt es immer häufiger zu „Datenpannen“, bei denen die Datensicherheit verletzt wird (z.B. [Bac10, Bra10]). Die ständig steigende Menge elektronisch gesammelter Information und die zunehmende kooperative Vernetzung der Daten untereinander führen vermehrt zu Problemen im Bereich der Datensicherheit. Hinzu kommen die steigende Komplexität größerer Systeme und der Umstand, dass dadurch im Falle einer Verletzung der Datensicherheit auch der verursachte Schaden wächst. Dies zeigt sich z.B. beim aufgedeckten Fall des Datendiebstahls von Kreditkartennummern [Spi09a] (Verletzung der Vertraulichkeit der Daten). Allein die Kosten der Umtauschaktion für die betroffenen Karten beliefen sich auf Millionenhöhe, vom Imageschaden für die beteiligten Firmen und den Missbrauchsschäden ganz zu schweigen. Weitere Beispiele für Gefährdungen im Bereich Datensicherheit sind die Verletzung der Vertraulichkeit der Daten beim sozialen Netzwerk SchülerVZ [Kre09, Sch] sowie Fälle von unrechtmäßigem Verkauf von Kundendaten [Spi09b].

Die Problematik der Datensicherheit verschärft sich darüber hinaus durch den Trend der Speicherung von Daten im Internet. Internetbasierte, einfach zu nutzende kooperative Web 2.0 Anwendungen [O'R05] wie Facebook [Fac], myspace [MyS] oder Flickr [Fli] werden immer populärer. Diese Anwendungen speichern auf zentralen Servern große Mengen persönlicher Daten der Nutzer. Ein einziges Sicherheitsproblem in einer dieser Anwendungen erhöht die Wahrscheinlichkeit von Schäden bereits beträchtlich.

Der Schutz persönlicher Daten vor unbefugten Zugriffen erfolgt maßgeblich durch Zugriffskontrolle. Durch die zunehmende Menge elektronisch gespeicherter persönlicher Daten und deren Verknüpfung untereinander wachsen die Anforderungen an die Zugriffskontrolle stetig.

1.1 Motivation

Das Aufkommen von P2P-Netzwerken verändert die Anforderungen an Datenbanken und ihre Zugriffskontrolle grundlegend [MS03]. Ihre hohe Skalierbarkeit, Ausfallsicherheit und Selbstorganisation erreichen diese Netzwerke unter anderem durch den vollständigen Verzicht auf eine zentrale Komponente. Diese positiven Eigenschaften der P2P-Netzwerke werden durch die in der Datenbankforschung entwickelten P2P-Datenbanken genutzt. P2P-Datenbanken können als Teilbereich der verteilten Datenbanken angesehen werden. Ziel von P2P-Datenbanken ist es, verschiedene Datenbanken bzw. deren Inhalte miteinander zu verknüpfen und so eine einheitliche Sicht auf die im Netzwerk publizierten Daten zu erhalten.

Das Einsatzgebiet von P2P-Datenbanken ist vielfältig, jedoch liegt ihr Schwerpunkt auf Anwendungen, bei denen ein zentraler Koordinator unerwünscht ist. Dies ist in bestimmten Bereichen von virtuellen Organisationen, bei wissenschaftlichen Kooperationen und bei der Zusammenarbeit zwischen Staaten der Fall. Hier wird eine Einschränkung der eigenen Souveränität selten akzeptiert. Ebenso sind Notfallsituationen, in denen eine rasche und flexible Kooperation zwischen den Beteiligten nötig ist, ein mögliches Einsatzszenario.

Problematisch an P2P-Datenbanken ist jedoch der Umstand, dass das beschriebene hohe Maß an Skalierbarkeit, Ausfallsicherheit und Selbstorganisation durch einen Verlust an Sicherheit relativiert wird, da die Frage der Zugriffskontrolle in P2P-Datenbanken in der Forschung bisher nur unzulänglich gelöst wurde. Ihre Aufgabe wäre es, den Zugriff auf gespeicherte Daten nur Benutzern zu gewähren, die gemäß den zuvor festgelegten Zugriffskontrollregeln autorisiert sind. Dies geschieht mittels differenzierter Zugriffskontrollmodelle mit verschiedenen Varianten von Zugriffskontrollregeln [dVFS07]. Die unterschiedlichen Modelle bedingen eine unterschiedliche Auswertung der Zugriffskontrollregeln durch die Zugriffskontrolle. Daneben gibt es verschiedene Möglichkeiten, die Durchsetzung dieser Zugriffskontrollregeln sicherzustellen. Neben der klassischen serverseitigen Durchsetzung, bei der die Zugriffskontrolle des Servers über die Herausgabe der Daten wacht, sind auch andere Arten denkbar, wie sie z.B. von Miklau und Suciu [MS03] vorgeschlagen werden.

Eine ideale Zugriffskontrolle für P2P-Datenbanken entspräche in ihrer Funktionalität jener für verteilte Datenbanken. Wichtigste Eigenschaft dieser Zugriffskontrolle ist der transparente Datenzugriff auf die Komponentendatenbanken, obwohl diese ihrerseits unterschiedliche Zugriffskontrollmechanismen einsetzen. Für P2P-Datenbanken ist die Verwendung dieser bereits etablierten Zugriffskontrollmechanismen der verteilten Datenbanken (siehe z.B. [dVS96]) nicht möglich. Sie benutzen einen zentralen Koordinator, der bei P2P-Datenbanken so nicht realisierbar ist. Zugriffskontrollmechanismen für P2P-Netzwerke können ebenfalls nicht ohne weiteres auf P2P-Datenbanken übertragen werden. Keiner der vorhandenen Ansätze integriert bestehende lokale Zugriffskontrollkomponenten.

Ohne eine Zugriffskontrolle bleibt das Einsatzgebiet von P2P-Datenbanken auf die Verarbeitung von nicht sicherheitskritischen Daten beschränkt, denn jedem Teilnehmer sind alle Daten im Netzwerk zugänglich. Eine Verarbeitung sicherheitsrelevanter oder persönlicher Daten ist auf diese Weise nicht möglich.

1.2 Ziele und Voraussetzungen der Arbeit

Ziel dieser Arbeit ist es zu beweisen, dass die Etablierung einer Zugriffskontrolle mit gleichwertiger Funktionalität wie jene föderierter Datenbanken ohne zentralen Koordinator für P2P-

Datenbanken möglich ist. Hierzu wird das neuartige Peer Access Control System (PACS) entwickelt und auf seine Effizienz und Umsetzbarkeit hin kritisch betrachtet und evaluiert.

Ausgehend vom aktuellen Stand der Forschung müssen hierzu folgende Fragen beantwortet werden:

- Kann die zentrale Autorität der globalen Zugriffskontrolle durch dezentrale Mechanismen ersetzt werden?
- Können bestehende lokal vorhandenen Zugriffskontrollregeln der einzelnen Benutzer zur Reduzierung des Einrichtungsaufwandes für eine globale Zugriffskontrolle genutzt werden?
- Ist der Einsatz erweiterter Zugriffskontrollmodelle analog zu zentralistischen Lösungen möglich?
- Wie müssen diese Privilegien verwaltet, gespeichert und durchgesetzt werden?
- In wie weit ist ein solcher Zugriffskontrollmechanismus ohne zentrale Autorität gegenüber möglicher im Netzwerk befindlicher bösartiger Knoten fehlertolerant? Wo liegen die Grenzen?
- Kann die Verfügbarkeit der Daten durch Datenreplikation garantiert werden, ohne deren Vertraulichkeit zu verletzen?

Die Klärung dieser Fragen erfolgt basierend auf der Annahme, dass die Teilnehmer die Vorteile eines P2P-Netzwerks mit gesteigerten Sicherheitsbedürfnisse kombinieren möchten. Für diese zahlt sich die Etablierung einer Zugriffskontrolle und damit der Aufwand zur Definition von Zugriffsregeln aus, da die Zugriffskontrolle die Einstiegshürde zum Beitritt zu einem solchen Netzwerk erhöht. Derartige Anfangsinvestitionen werden nur von Teilnehmern getragen, die länger im Netzwerk verbleiben wollen. Hieraus resultiert die Annahme einer verminderten Dynamik für die in dieser Arbeit betrachteten P2P-Systeme.

Basierend auf bisherigen Forschungsergebnissen finden zwei Anwendungsgebiete Berücksichtigung. Ein erster Anwendungsbereich stellen P2P-Datenbanken dar, bei denen es sich um P2P-Netzwerk handelt, dessen Peers Datenbankserver sind. Die Teilnehmer besitzen deshalb eine hohe Verfügbarkeit. Zudem handelt es sich hierbei um kleine Gruppen mit weniger als 1000 Teilnehmern. Die einzelnen Peers bedienen mehrere lokale Benutzer und verfügen schon deshalb über ausgefeilte lokale Zugriffskontrollkomponenten.

Das zweite Anwendungsgebiet bedient kontrollierte P2P-Netzwerke für den Datenaustausch. Diese Netzwerke sind durch eine größere Anzahl (> 1000) an Teilnehmern gekennzeichnet. Auch wenn deren Teilnehmer verlässliche Verfügbarkeitszeiten besitzen, sind sie möglicherweise zeitweise nicht erreichbar. Dennoch handelt es sich um Teilnehmer, die sich über längere Zeit regelmäßig an diesem Netzwerk beteiligen und entsprechend die erhöhten Eintrittsinvestitionen in Kauf nehmen. Teilnehmer können ebenfalls lokale Zugriffskontrollkomponenten und mehrere Benutzer besitzen.

Der Hauptunterschied zwischen den beiden Anwendungsgebieten besteht darin, dass beim Netzwerk aus Servern keine Datenreplikation der im Netzwerk publizierten Daten stattfindet. Entsprechend sind die Daten eines Peers nur verfügbar, solange sich dieser im Netzwerk befindet. Beim zweiten Anwendungsgebiet findet hingegen aufgrund der unbeständigeren Verfügbarkeitszeiten der Teilnehmer eine Datenreplikation statt. Die Daten eines Peers bleiben also im Netzwerk verfügbar, auch wenn dieser das Netzwerk kurzzeitig verlässt.

1.3 Eigene Beiträge

Die Entwicklung der dezentralen Zugriffskontrolle PACS mit Unterstützung erweiterter Zugriffsmodelle, wie RBAC oder administrativer Delegation, umfasst folgende Beiträge.

- Die Koordination der bestehenden heterogenen Zugriffskontrollmodelle durch Einführung von XACML Policies. Neben den hierfür nötigen Annotationen ist hier insbesondere ihre dezentrale Koordination zu nennen. Diese erfolgt durch Erteilung globaler Privilegien, die wiederum dezentral verwaltet werden müssen.
- Die Einführung eines Zugriffskontrollmodells, das den Anforderungen entspricht und dennoch dezentral verwaltet werden kann.
- Der Entwurf einer dezentralen Verwaltung erweiterter Zugriffskontrollmodelle, die mit bisherigen Ansätzen nicht möglich war.
- Die Entwicklung des für die Verwaltung der Zugriffskontrollmodelle notwendigen dezentralen Privilegienspeichers. Hierfür wurde sowohl die Nutzung einer DHT als auch eines unstrukturierten Netzwerks untersucht.
  - Für die DHT sind hierfür Erweiterungen des zugrundeliegenden Chord Protokolls und der DHT-Anwendung selbst erforderlich, um eine verlässlichen DHT garantieren zu können.
  - Beim unstrukturierten P2P-Netzwerk erfolgt die Einführung einer aktiven Replikation mit definierter Replikationsgruppe. Erst hierdurch ist eine verlässliche Speicherung von Daten im unstrukturierten Netzwerk möglich.
- Schutz der Integrität der im Privilegienspeicher verwalteten Privilegien.
- Dezentrale Durchsetzung des Zugriffskontrollmodells mit administrativer Delegation und RBAC.
  - Für die serverseitige Durchsetzung wird die Organisation der Privilegien im Privilegienspeicher, und damit der Ablauf der Durchsetzung, festgelegt.
  - Die ebenfalls entwickelte clientseitige Durchsetzung stellt mit ihrer Unterstützung für RBAC und administrative Delegation ein Novum für clientseitige Durchsetzungsvarianten dar. Die Kombination zwischen gruppenbasierter und clientseitiger Durchsetzung und die autonome Verwaltung von Gruppen sind dabei die entscheidenden Beiträge. Hierdurch wird es erstmals möglich, moderne Zugriffskontrollmodelle auch clientseitig durchzusetzen und somit eine sichere Replikation der Daten zu ermöglichen.

1.4 Aufbau der Arbeit

Zu Beginn werden in Kapitel 2 für die Arbeit wichtige Begriffe definiert und der späteren Lösung zugrundeliegende Technologien eingeführt. Hier erfolgt auch die Darstellung des aktuellen Forschungsstands im Bereich P2P-Datenbanken. Daneben wird für verwandte Ansätze und Forschungsresultate diskutiert, in wieweit sie die gestellten Anforderungen erfüllen.

Kapitel 3 ist das Hauptkapitel dieser Arbeit. Es erläutert Aufbau und Bestandteile von PACS. Nach einer Übersicht über Funktionalität und Komponenten der Zugriffskontrolle wird auf die einzelnen Bestandteile genauer eingegangen. Nach der Beschreibung der Authentifizierung von Benutzern in PACS wird das verwendete Zugriffskontrollmodell erläutert. Hier wird auf die Besonderheit der lokalen und globalen Ebene mit ihren unterschiedlichen Zugriffskontrollmodellen eingegangen. Es folgt die Betrachtung der Privilegienverwaltung.

Der verteilte Privilegienspeicher wird beschrieben. Die Anforderungen und Bedrohungen werden definiert. Die Schutzmechanismen zur Vermeidung von Manipulationen gespeicherter Privilegien werden erläutert. Daran schließen sich die Ausführungen zu den beiden für PACS entwickelten Privilegienspeichern an, dem auf einem unstrukturierten P2P-Netzwerk basierenden Privilegienspeicher und dem DHT-basierten Privilegienspeicher.

Nach der Erläuterung des Privilegienspeichers erfolgt die Darstellung der serverseitigen und clientseitigen Durchsetzungsvarianten, die die Einhaltung der gemäß dem Zugriffskontrollmodell erstellten Zugriffsregeln kontrollieren.

In Kapitel 4 werden die Simulationsumgebung, deren Aufbau und ihre Einstellungen erläutert. Anschließend werden die Besonderheiten und Einstellungen der Simulation für den DHT-basierten Privilegienspeicher und den auf einem unstrukturierten P2P-Netzwerk basierenden Privilegienspeicher dargelegt. Am Ende wird die Arbeitslast der Simulation genauer erläutert.

In Kapitel 5 erfolgt die Evaluation der zuvor in Kapitel 3 beschriebenen PACS-Zugriffskontrolle. Schwerpunkt ist der Privilegienspeicher mit seinen Bestandteilen. Hier erfolgt die Präsentation der Simulationsergebnisse zur Arbeitsweise des Chord Protokolls und den daran vorgenommenen Erweiterungen, sowie zur Zuverlässigkeit des DHT-basierten Privilegienspeichers. Die Funktionsweise des unstrukturierten P2P-Netzwerks wird durch Simulationsergebnisse dargestellt. Die Zuverlässigkeit und der Aufwand des zweiten, auf dem unstrukturierten P2P-Netzwerks basierenden Privilegienspeichers, wird ebenfalls anhand von Simulationen bewiesen. Zudem enthält dieses Kapitel eine Komplexitätsanalyse der clientseitigen Durchsetzung von PACS, sowie eine Aufwandsabschätzung von PACS bei Verwendung der beiden Privilegienspeicher und Durchsetzungsvarianten.

In Kapitel 6 werden die Resultate zusammengefasst und diskutiert sowie ein Ausblick auf zukünftige Forschungsarbeiten gegeben.

Kapitel 2

Grundlagen

Dieses Kapitel definiert und erläutert in den Sektionen 2.1 und 2.2 wichtige Grundbegriffe. Daran anschließend erfolgt eine kurze Einführung in Technologien, auf denen diese Arbeit basiert. In Kapitel 2.3 werden kryptographischen Methoden vorgestellt die für Umsetzung der Zugriffskontrolle in Kapitel 3 benötigt werden. Ihr Verständnis ist für die Funktionsweise dieser Zugriffskontrolle notwendig und wird deshalb hier etwas genauer erläutert. Kapitel 2.4 gibt einen Überblick über die verschiedene existierenden Zugriffskontrollmodelle. Die Einhaltung der Zugriffskontrollregeln (Durchsetzung) in diesen Modellen erläutert Kapitel 2.5. Nach einer kurzen Zusammenfassung zu Föderierten Datenbanken gibt Kapitel 2.7 eine Einführung in Peer-to-Peer (P2P) Systeme, gefolgt von einer Beschreibung aktueller Forschungsprojekte in P2P-Datenbanken. Einen Überblick über die aktuelle Forschung im Bereich Sicherheit in P2P-Systemen liefert Kapitel 2.9. Zum Abschluss werden die existierenden Ansätze miteinander verglichen und ihre Anwendungsmöglichkeit auf den hiesigen Anwendungsbereich untersucht (Kapitel 2.10).

2.1 Definition Datensicherheit

Datensicherheit basiert auf den folgenden drei Grundpfeilern [EN07]:

Vertraulichkeit Die Vertraulichkeit der gespeicherten Daten gewährleistet, dass diese sicher aufbewahrt und nur an berechtigte Benutzer freigegeben werden.

Verfügbarkeit Die Verfügbarkeit garantiert, dass gespeicherte Daten über einen zugesicherten Zeitraum den berechtigten Personen in der erforderlichen Qualität zugänglich sind.

Integrität Datenintegrität ist der Schutz der gespeicherten Daten vor unerlaubten Änderungen. Dies verhindert auch die unerlaubte Löschung von Daten.

Die Gewährleistung der Datensicherheit erfolgt entsprechend der drei Grundpfeiler auf unterschiedliche Art und Weise. Die Vertraulichkeit wird durch Verschlüsselungstechniken und durch die Beschränkung des Datenzugriffs gewährleistet. Die Verfügbarkeit kann durch Replikation (Anfertigen von identischen Kopien der Daten und Verteilung dieser Kopien auf verschiedene Orte) erhöht werden. Daneben lassen sich Maßnahmen zum konkreten Erkennen und Abwehren von Denial of Service Angriffen nennen. Die Integrität der Daten wird durch die Beschränkung des Datenzugriffs und Signatur der Daten erreicht. Der Verlust von Daten kann durch Datensicherung (Backup) vermieden werden.

Datensicherheit kann von einem System nur so lange sichergestellt werden, wie die Daten innerhalb seines Kontrollbereichs bleiben. Sobald sich Daten außerhalb des Systems befinden (z.B. eine Datei wird von einem Benutzer heruntergeladen), hat es keinerlei Kontrolle mehr über die Daten und kann deshalb deren Sicherheit nicht mehr garantieren. Der Kontrollbereich eines Systems kann durch sogenannte Trusted Computing Technology (TCT) [Tru09] auch auf den Client ausgedehnt werden. Dabei handelt es sich aber um einen ganz eigenen Themenbereich, der sich außerhalb dieser Arbeit befindet.

In P2P-Systemen spielt auch das Vertrauen zwischen zwei Einheiten eine Rolle. Für diese Arbeit soll die Definition nach Olmedilla u.a. [ORMN05] gelten: Vertrauen eines Peers A in einen Peer B zur Erfüllung einer Aufgabe X ist der messbare Glaube von A, dass B innerhalb der festgelegten Zeit und des gegebenen Kontextes, die Aufgabe X zuverlässig ausführt. Übliche Grundlagen dieses Glaubens sind z.B. Wissen, Erfahrungen aus der Vergangenheit oder auch Empfehlungen anderer.

2.2 Sicherheit von Datenbanken

Datenbank Management Systeme (DBMS) sind häufig Speicherort sensibler, schutzbedürftiger Daten. Insofern ist es wenig verwunderlich, dass Datenbanken verschiedene Mechanismen besitzen, um die Datensicherheit der in ihnen gespeicherten Daten zu garantieren. Neben der Transaktionsverwaltung und der Wiederherstellungskomponente erfolgt die Wahrung der Datensicherheit in einem DBMS durch die Schutzkomponenten Inferenzkontrolle, Datenflusskontrolle, Datenverschlüsselung und Zugriffskontrolle. Diese Komponenten werden zusammengefasst als Datenbanksicherheits- und Autorisierungssubsystem bezeichnet [EN07].

- Die *Inferenzkontrolle* versucht zu verhindern, dass auf geschützte Information aufgrund anderer Informationen geschlossen werden kann, obwohl der Zugriff auf diese geschützten Informationen nicht erlaubt ist. Dies ist insbesondere bei aggregierten Daten möglich, wie sie in Statistikdatenbanken vorliegen.
- Die *Datenflusskontrolle* soll verhindern, dass Daten an unautorisierte Nutzer weitergeleitet werden.
- Die *Datenverschlüsselung* garantiert die Vertraulichkeit der Daten durch geeignete Verschlüsselungstechniken (siehe Kapitel 2.3).
- Die *Zugriffskontrolle* ist Thema dieser Arbeit. Sie wird ausführlich im nächsten Kapitel besprochen.

2.2.1 Zugriffskontrolle

Die Zugriffskontrolle stellt den wichtigsten Teil der Sicherheitsmechanismen von Datenbanken dar. Die Aufgabe der Zugriffskontrolle ist es erstens, jeden Zugriff auf Daten des Systems zu kontrollieren und zweitens, sicherzustellen, dass ausschließlich autorisierte Zugriffe auf die Daten des DBMS möglich sind [dVFS07]. Die Zugriffskontrolle untergliedert sich in drei Ebenen [dVFS07, FKC07]:

Zugriffskontrollregeln (aufgrund ihrer englischen Bezeichnung oft auch kurz *Policies* genannt) sind allgemeine Regeln, wie und was durch die Zugriffskontrolle geschützt werden

soll. Dieses Regeln sind häufig Änderungen unterworfen, da sie an die sich ändernde Geschäftsprozesse und gesetzliche Regelungen angepasst werden müssen.

Zugriffskontrollmodelle sind formale Repräsentationen der Zugriffskontrollregeln und ihrer Arbeitsweise.

Zugriffskontrollmechanismen sind Implementierungen der Kontrollfunktionen, die durch die Zugriffskontrollregeln festgelegt werden.

Ein DBMS ist üblicherweise für einen Mehrbenutzerbetrieb konzipiert, weshalb eine Benutzerverwaltung inhärenter Bestandteil der Zugriffskontrolle ist. Dabei ist es wichtig zwischen Autorisierung und Authentifizierung zu unterscheiden [FKC07].

Authentifizierung Die Authentifizierung ist die eindeutige Zuordnung einer Person zu einem Benutzer der Datenbank. Hierzu muss die Person einen Beweis liefern, dass sie berechtigt ist, als entsprechender Benutzer mit der Datenbank zu arbeiten. Übliche Verfahren sind die Prüfung von Benutzername und Passwort oder der Nachweis des Besitzes eines privaten Schlüssels. Neue Verfahren sind Fingerabdruck- und Irisscan. Authentifizierung ist die Grundlage für die Überprüfung der Autorisierung.

Autorisierung Bezeichnet einen erlaubten Zugriff eines Benutzers auf ein Objekt eines Systems. Die Autorisierungen eines Benutzers ist die Menge aller Zugriffe die dem Benutzer erlaubt sind. Besitzt ein Benutzer die Erlaubnis eine Aktion a durchzuführen, dann (und nur dann) gilt der Benutzer für a als autorisiert.

Jeder Benutzer der Datenbank muss sich zunächst erfolgreich gegenüber dem DBMS authentifizieren. Nach erfolgreicher Anmeldung erfolgt dann die entsprechende Prüfung der Autorisierung für die vom Benutzer angeforderten Aktionen.

2.3 Kryptographie

Durch Kryptographie, d.h. der Verschlüsselung von Informationen, kann die Vertraulichkeit von Daten sichergestellt werden. Weitere Anwendungsbereiche von Verschlüsselung sind die abhörsichere Kommunikation über offene Kommunikationsnetze und die Authentifizierung von Benutzern. Zu den Grundbegriffen:

Klartext m Ein Klartext (im Englischen „Plaintext“) Bezeichnet die unverschlüsselten Daten, die als Eingabe für den Verschlüsselungsalgorithmus dienen.

Verschlüsselungsalgorithmus E Ein Verschlüsselungsalgorithmus ist ein Algorithmus, der ausgeklügelte Transformationen des Klartextes vornimmt. Als Eingabe dienen ihm hierbei der Klartext und ein Schlüssel.

Chiffretext c Ein Chiffretext (im Englischen „Ciphertext“) ist das Ergebnis einer Klartexttransformation durch einen Verschlüsselungsalgorithmus.

Entschlüsselungsalgorithmus D Ein Entschlüsselungsalgorithmus produziert aus Schlüssel und Chiffretext wiederum den Klartext.

2.3.1 Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung (auch Kryptographie mit privaten Schlüsseln genannt) wird zunächst der private Schlüssel zwischen den Kommunikationspartnern ausgetauscht. Dieser Austausch des privaten Schlüssels $k \in \mathcal{K}$, mit $\mathcal{K}$ als Menge der Schlüssel, muss über einen sicheren Kanal erfolgen. Anschließend lässt sich durch den Verschlüsselungsalgorithmus $E : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$ der entsprechend Chiffretext $c = E_k(m)$ berechnen, wobei $\mathcal{M}$ die Menge der Klartexte und $\mathcal{C}$ die Menge der Chiffretexte ist. Der Chiffretext c kann dann über das unsichere Netzwerk versandt werden. Der Empfänger wendet den entsprechenden Entschlüsselungsalgorithmus $D : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$ an, und erhält so den Klartext $m = D_k(c)$. Es gibt verschiedenste symmetrische Verschlüsselungsalgorithmen mit bewiesener Sicherheit. Die bekanntesten sind der Data Encryption Standard (DES) [Nat99], der Advance Encryption Standard (AES) [DR02] und der International Data Encryption Algorithm (IDEA) [LM91].

Allgemein sind symmetrische Verschlüsselungsverfahren bei gleicher Sicherheit etwa 1000-mal weniger rechenintensiv (und damit schneller) als die im Folgenden vorgestellten asymmetrischen Verschlüsselungsverfahren [Sch96]. Der Austausch des privaten Schlüssel über einen sicheren Kanal ist allerdings problematisch. In der Praxis werden für den Austausch der privaten Schlüssel häufig asymmetrische Verschlüsselungsverfahren verwendet. Dies sind hybride Verschlüsselungssysteme.

2.3.2 Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung (auch Kryptographie mit öffentlichen Schlüsseln genannt) werden zwei verschiedene Schlüssel verwendet. Ein Schlüssel ist öffentlich (daher auch der Name des Verschlüsselungsverfahrens), der andere geheim. Wichtig hierbei ist, dass der geheime Schlüssel (auch privater Schlüssel genannt) nicht aus dem öffentlichen Schlüssel berechnet werden kann. Um eine verschlüsselte Kommunikation zwischen den zwei Teilnehmern Alice und Bob aufzubauen, muss jeder der beiden zunächst für sich selbst sowohl einen privaten K^{-1} als auch einen öffentlichen Schlüssel K erzeugen, wobei $(K^{-1}, K) \in \mathcal{K}$. Die öffentlichen Schlüssel werden publiziert, sodass Alice den öffentlichen Schlüssel von Bob kennt und umgekehrt. Will nun Alice Daten verschlüsseln, die lediglich von Bob gelesen werden können sollen, verwendet sie den Verschlüsselungsalgorithmus $E : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$ mit dem öffentlichen Schlüssel von Bob $c = E_K(m)$. Bob kann nach der Übertragung an ihn die empfangenen Daten durch den Entschlüsselungsalgorithmus $D : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$ entschlüsseln, indem er seinen privaten Schlüssel verwendet. Er erhält somit den Klartext $m = D_{K^{-1}}(c)$. Dies ist möglich, da für asymmetrische Verschlüsselungsverfahren gelten muss, dass $D_{K^{-1}}(E_K(m)) = m$. Das bekannteste asymmetrische Verschlüsselungsverfahren ist RSA [RSA78].

Die Benutzer eines solchen Systems müssen jeweils Zugang zum entsprechenden authentischen öffentlichen Schlüssel der anderen Nutzer besitzen. Dies wird durch eine Zertifizierungsstelle gewährleistet, die die öffentlichen Schlüssel signiert und damit zertifiziert. Sie ist Bestandteil einer Public Key Infrastruktur (PKI). Nur so ist sichergestellt, dass die verschlüsselten Daten auch vom richtigen Benutzer gelesen werden können.

2.3.2.1 Digitale Signaturen

Durch die Verschlüsselung wird die Vertraulichkeit der Daten gewährleistet. Die Integrität der Daten kann hingegen durch digitale Signaturen sichergestellt werden. Digitale Signaturen

basieren auf der Berechnung eines eindeutigen Hashwerts. Die Hashfunktion ist eine Einwegfunktion, die für beliebig große Zeichenmengen immer gleich lange Zeichenketten erzeugt, auch Hashwerte genannt. Durch die Eigenschaft der Einwegfunktion ist sichergestellt, dass aus dem berechneten Hashwert keine Rückschlüsse auf die ursprüngliche Zeichenmenge möglich sind. Zudem sollte die Hashfunktion kollisionsresistent sein, d.h. es sollte unmöglich sein, zwei unterschiedliche Zeichenketten zu finden, die denselben Hashwert besitzen. Die bekanntesten kollisionsresistenten Hashfunktionen sind MD4 [Riv91], MD5 [Riv92] (mittlerweile gebrochen) und SHA-1 als Bestandteil des Secure Hash Standards [Nat08].

Die Verschlüsselung eines Hashwerts mit dem privaten Schlüssel von Alice, wird als digitale Signatur von Alice bezeichnet. Der Signaturalgorithmus $SIG : \mathcal{K} \cdot \mathcal{M} \rightarrow \mathcal{S}$ kombiniert den privater Schlüssel $K^{-1} \in \mathcal{K}$ des Unterschreibenden und das zu unterschreibende Datenobjekt $m \in \mathcal{M}$ zu einer Signatur $s = SIG(K^{-1}, m) = SIG_{K^{-1}}(m) = E_{K^{-1}}(h(m))$, wobei h hier die sichere d.h. kollisionsresistente Hashfunktion bezeichnet. Eine solche Signatur kann überprüft werden, indem die entschlüsselte Signatur mit dem Hashwert des Datenobjektes verglichen wird, d.h. ob $D_K(s) \equiv h(m)$, wobei K dem öffentlichen Schlüssel des Unterzeichners und h wiederum der Hashfunktion entspricht. Wurde das Datenobjekt nicht verändert, ist diese Verifikation erfolgreich, da $D_K(E_{K^{-1}}(h(m))) = h(m)$. Durch eine solche digitale Signatur kann zweifelsfrei festgestellt werden, dass der Besitzer eines bestimmten privaten Schlüssels das Datenobjekt signiert hat. Durch Verwendung kollisionsresistenter Hashfunktionen ist es nahezu ausgeschlossen, dass Änderungen an dem Datenobjekt vorgenommen werden können, ohne dass sich die Signatur dabei ändert. Die bekanntesten Algorithmen zur Erzeugung digitaler Signaturen sind RSA [RSA78] und DSA [Nat09].

2.3.2.2 Zertifikate

Zertifikate sind digital signierte öffentliche Schlüssel. Neben der Signatur und dem Schlüssel enthalten sie zwingend einen eindeutigen Benutzernamen und zumeist weitere Angaben zum Eigentümer und zur Gültigkeit des öffentlichen Schlüssels. Daneben enthält das Zertifikat noch Informationen zur Zertifizierungsstelle. X.509 [Tel05] hat sich als Standard für Zertifikate durchgesetzt.

Zertifikate werden von einer Zertifizierungsinstanz ausgestellt und enthalten deshalb deren Signatur [Sch96]. Durch die Signatur bürgt die Zertifizierungsstelle für die Authentizität des dort enthaltenen öffentlichen Schlüssels. Zudem veröffentlicht die Zertifizierungsstelle eine Liste widerrufener Zertifikate (certificate revocation list), um im Falle eines Missbrauchs oder des Verlusts des zugehörigen privaten Schlüssels das Zertifikat als ungültig erklären zu können. Bei der Prüfung der Korrektheit eines Zertifikats muss in dieser Liste nachgesehen werden, ob das zu überprüfende Zertifikat dort enthalten ist. Daneben ist es nötig, die Signatur der Zertifizierungsstelle zu überprüfen. Dies kann nur durch den öffentlichen Schlüssel der Zertifizierungsstelle geschehen. Dieser Schlüssel ist im sogenannten Root-Zertifikat gespeichert. Auf die Authentizität dieses Zertifikats müssen alle Teilnehmer der PKI vertrauen. Es gibt auch Ansätze, Zertifikate um weitere Attribute anzureichern, um z.B. auch Privilegien dort zu hinterlegen. Diese Zertifikate heißen Attributzertifikate [Far02, PS00].

2.3.2.3 Kryptographische Authentifizierung von Benutzern

Die Authentifizierung unter Zuhilfenahme von Verschlüsselung ist auch unter den Fachbegriffen Challenge-Response Authentifizierung oder Handshaking-Protokoll bekannt [JP03]. Die

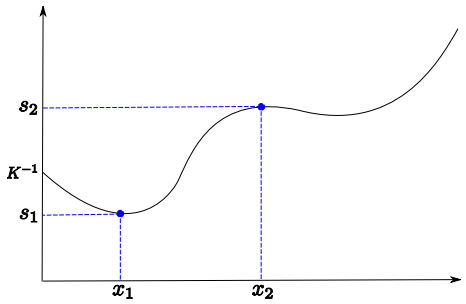


Abbildung 2.1: Erzeugung von Teilschlüssel für Schlüssel K^{-1}

Authentifizierung zwischen zwei Benutzern erfolgt nicht durch den Austausch von Passwörtern, sondern durch korrekte Antworten auf zufällige Prüfungen (sogenannte „Challenges“). Ist eine PKI vorhanden, kann die Authentifizierung zweier Benutzer Alice und Bob wie folgt geprüft werden. Es sei K_B^{-1} der private Schlüssel von Bob und K_B der zugehörige öffentliche Schlüssel. Alice authentifiziert Bob nun, indem sie überprüft ob Bob im Besitz des Schlüssels K_B^{-1} ist. Alice wählt hierzu einen zufälligen Klartext m_A und verschlüsselt ihn mit dem öffentlichen Schlüssel von Bob, $c_A = E_{K_B}(m_A)$. Nur Bob kann diesen so verschlüsselten Chiffretext c_A entschlüsseln und m_A an Alice zurücksenden. Ist dies der Fall, akzeptiert Alice, dass sie mit Bob kommuniziert.

2.3.3 Schwellenkryptographie

Bei Schwellenkryptographie (im Englischen „Threshold Cryptography“) handelt es sich um einen Teilbereich der gruppenorientierten Kryptographie [JP03]. Ist zum erfolgreichen Abschluss eines kryptographischen Vorgangs (Signieren, Verschlüsseln und Entschlüsseln) eine Zusammenarbeit von t Benutzern aus insgesamt n Teilnehmern erforderlich, ist die Rede von einer (t, n) Schwellenkryptographie. Wichtig ist hierbei, dass jede Teilmenge der Größe t $(t \leq n)$ diese Operationen durchführen kann. Teilmengen von n der Größe g $(0 \leq g \leq (t-1))$ können aufgrund des zugrundeliegenden Algorithmus keinerlei Operationen durchführen. Grundlage der Schwellenkryptographie ist die Aufteilung eines privaten Schlüssels auf alle Teilnehmer n.

2.3.3.1 Teilschlüssel Erzeugung und Verifikation

Die Verteilung des privaten Schlüssels K^{-1} erfolgt durch Anwendung von Shamir's (t, n) Secret Sharing Technique [Sha79]. Hierbei gilt, dass t Teilnehmer den Schlüssel wieder erzeugen können aber $t-1$ keinerlei Informationen über den privaten Schlüssel erhalten. Hierzu wird ein beliebiges Polygon $f(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$ des Grades $(t-1)$ gewählt, so dass $K^{-1} = f(0)$. Mit diesem Polynom werden nun Teilschlüssel $s_i = f(x_i)$ erzeugt und auf einem sicheren Übertragungsweg an die Teilnehmer $\mathcal{P}_i (i = 1, \ldots, n)$ verteilt. Die x Werte werden hierbei zufällig ausgewählt und anschließend veröffentlicht. Die Erzeugung der Teilschlüssel s_1 und s_2 für die x Werte x_1 und x_2 ist in Abbildung 2.1 graphisch dargestellt.

Beliebige t Teilnehmer können nun durch Austausch ihrer Teilschlüssel t Punkte des geheimen Polynoms erhalten. Durch die Lagrange Interpolation kann aus den t Punkten das Polynom $f(x)$ des Grades $(t-1)$ bestimmt werden. Da $K^{-1} = f(0)$ gilt, kann sodann der private Schlüssel K^{-1} zurückgewonnen werden.

Um verlorene Teilschlüssel auch ohne das Eingreifen des Erzeugers rekonstruieren zu können, werden die einzelnen Teilschlüssel erneut durch Anwendung von Shamir's (t, n) Secret Sharing Technique auf andere Teilnehmer verteilt. Der Erzeuger bildet hierbei für jeden Teilschlüssel s_i ein neues Shamir's Secret Sharing Schema. Der Ablauf ist wie folgt:

1. Für jeden Teilnehmer d_i nimmt der Erzeuger den Teilschlüssel s_i des Teilnehmers und wendet das (t, n) Secret Sharing Schema dafür an. Das Polynom $f_i(x)$ ist vom Grad $(t-1)$ und $f_i(0) = s_i$.

2. Der Erzeuger berechnet diese Funktion $f_i(x), x_i \in \mathcal{R}$ nun für n Punkte $[x_1, \ldots, x_n]$ und erhält als Resultat n Teilteilschlüssel $[s_{i,1}, \ldots, s_{i,n}]$ (im Englischen „Share-Share“) mit $f_i(x_j) = s_{i,j}$.

3. Der Erzeuger übermittelt nun an jeden Teilnehmer d_i Teilteilschlüssel $[s_{1,i}, \ldots, s_{n,i}]$. Teilnehmer d_i erhält also die Teilteilschlüssel für den i-ten x Wert von jedem Teilteilschlüssel Polynom. Von jedem Teilteilschlüssel wird noch ein Hashwert erzeugt und veröffentlicht. Durch diesen kann die Korrektheit eines Teilteilschlüssels während der Teilschlüssel Wiederherstellung überprüft werden.

Um den Teilschlüssel von Teilnehmer d_j wiederherzustellen muss jeder verbleibende Teilnehmer d_i seinen Teilteilschlüssel $s_{i,j}$ zur Verfügung stellen. Sobald t korrekte Teilteilschlüssel gesammelt wurden (die Korrektheit der Teilteilschlüssel kann durch ihre publizierten Hashwerte überprüft werden), kann Teilschlüssel s_j durch Anwendung der Lagrange Interpolation wiederhergestellt werden.

Um die Vorteile von RSA mit Shamir's Schema kombinieren zu können, bedarf es einiger Anpassungen [DF92]. Da die RSA Berechnungen bei Ver- und Entschlüsselungen modulo N sind, wird N für deren Berechnung benötigt und deshalb mit dem öffentlichen Schlüssel K veröffentlicht. Des Weiteren müssen alle x_i $(i = 1, \ldots, n)$ ungerade sein. Dadurch ist es auch nötig, den privaten Schlüssel zu verschieben. Anstatt dem zuvor beschriebenen $f(0) = K^{-1}$, wird dieser bei RSA $f(-1) = K^{-1}$ gleichgesetzt.

2.3.3.2 Schwellenentschlüsselung

Der private Schlüssel wird über das zuvor beschriebene Shamir's Secret Sharing Schema auf n Teilnehmer verteilt. Die Verschlüsselung des Klartextes m erfolgt mit dem öffentlichen Schlüssel K, $c = E_K(m)$. Das besondere an der Schwellenverschlüsselung ist, dass die Teilnehmer ihren Teilschlüssel s_i wie einen richtigen privaten Schlüssel benutzen. Für die Entschlüsselung berechnet so jedes Mitglied $c_i \equiv D_{s_i}(c)$ und gibt dieses teilentschlüsselte Resultat c_i an einen Koordinator weiter. Der Koordinator sammelt die teilentschlüsselten Resultate $c_i, i = 1, \ldots, t$. Hierbei wird auch die Menge der Teilnehmer der Verschlüsselung $\mathcal{B} = \{P_1, \ldots, P_t\}$ bestimmt. Durch die nun bekannte Teilnehmermenge $\mathcal{B}$ können die teilentschlüsselten Resultate c_i durch den Koordinator korrigiert werden. Die Korrektur unterscheidet sich je nach gewählten Verschlüsselungsverfahren. Bei RSA besteht diese Korrektur aus $\hat{c}_i \equiv D_{\prod_{P_j \in \mathcal{P} \setminus \mathcal{B}; j \neq i}(x_i - x_j) \prod_{P_j \in \mathcal{B}; j \neq i}(-1 - x_j)}(c_i)$. Das vollständig entschlüsselte Datenobjekt ergibt sich dann durch Kombination der korrigierten Teilentschlüsselungen. Bei RSA geschieht die Kombination durch $\prod_{P_i \in \mathcal{B}} \hat{c}_i \cdot c \equiv m \pmod{N}$ [DF92]. Für andere Verschlüsselungsverfahren wie z.B. ElGamal [Gam85] gelten entsprechend andere Korrektur- und Kombinationsfunktionen [DF89].

2.3.3.3 Schwellensignatur

Es wird angenommen, dass alle Teilnehmer einen Teilschlüssel s_i des privaten Schlüssels K^{-1} besitzen. t der n Teilnehmer können zusammen eine Signatur $SIG_{K^{-1}}(m)$ für das Datenobjekt m erzeugen, ohne dass einer der Teilnehmer den privaten Schlüssel K^{-1} besitzt. Die so erzeugte Signatur kann durch Verwendung des öffentlichen Schlüssels K von jedem Teilnehmer überprüft werden (siehe Kapitel 2.3.2.1). Die Erzeugung einer Schwellensignatur für das Datenobjekt m erfolgt wie folgt: Jeder Teilnehmer signiert das Datenobjekt mit seinem Teilschlüssel s_i und erhält so Teilsignaturen $sig_i = SIG_{s_i}(m)$. Der Koordinator sammelt t dieser Teilsignaturen und bestimmt die Menge der Teilnehmer der Verschlüsselung $\mathcal{B} = \{P_1, \ldots, P_t\}$. Anschließend werden Teilsignaturen, wie die teilentschlüsselten Resultate bei der Schwellenentschlüsselung, vom Koordinator korrigiert. In RSA erfolgt dies durch $\hat{sig}_i \equiv E_{\prod_{P_j \in \mathcal{P} \setminus \mathcal{B}; j \neq i}(x_i - x_j) \prod_{P_j \in \mathcal{B}; j \neq i}(-1 - x_j)}(sig_i)$. Die Signatur kann durch $SIG_{K^{-1}}(m) = \prod_{P_i \in \mathcal{B}} \hat{sig}_i \pmod N$ dann zusammengesetzt werden.

Verifikation der Teilsignaturen Diese Verifikation der Teilsignaturen ist immer dann nötig, falls die durch zusammensetzten der Teilsignaturen erhaltene vollständige Signatur $SIG_{K^{-1}}$ ungültig ist. Dies ist schon bei einer fehlerhaften Teilsignatur der Fall. Durch die Möglichkeit die einzelnen Teilsignaturen verifizieren zu können ist es dann möglich die falsche Teilsignatur zu erkennen und auszutauschen.

Die Verifikation einer Teilsignatur erfolgt für RSA gemäß dem Protokoll von Gennaro u.a. [GRJK00]. Zur Verifikation dienen hierbei Teilsignatur $SIG_{s_i}(w)$ einer Musternachricht w. Die Teilsignaturen und die Musternachricht sind hierbei öffentlich zugänglich. Das Protokoll läuft zwischen dem Prüfer und dem zu untersuchenden Peer ab.

1. Der Prüfer hat eine Teilsignatur $SIG_{s_i}(m)$ der Nachricht m von Peer P_i erhalten.

2. Der Prüfer wählt zwei Zufallszahlen $i, j \in \{1, \ldots, N\}$. Anschließend berechnet er die Probe $r = m^i \times w^j \pmod N$ und sendet diese zu Peer P_i.

3. Peer P_i signiert die Probe r nun mit seinem Teilschlüssel $SIG_{s_i}(r)$ und sendet diese Signatur zum Prüfer zurück.

4. Der Prüfer verifiziert nun ob $SIG_{s_i}(r) = SIG_{s_i}(m)^i \times SIG_{s_i}(w)^j \pmod N$. Ist die Prüfung erfolgreich akzeptiert der Prüfer $SIG_{s_i}(m)$ als gültige Teilsignatur.

2.4 Zugriffskontrollmodelle

Zur nachfolgenden Spezifizierung von Zugriffskontrollsystemen und Privilegien wird, in Anlehnung an [FKC07, CFMS94], folgende Terminologie verwendet:

Benutzer Ein Benutzer ist eine natürliche Person, die mit dem Computer interagiert. Der Benutzer wird authentifiziert und dadurch auf ein Subjekt abgebildet.

Subjekt Ein Subjekt ist die aktive Einheit im System. Ein Subjekt repräsentiert immer genau einen Benutzer, aber für einen Benutzer können mehrere unterschiedliche Subjekte existieren.

Objekt Objekte entsprechen den Ressourcen des Computersystems (z.B. Datenbanken, Dateien, Datensätze in der Datenbank, usw.) die geschützt werden sollen.

Privileg Ein Privileg ist eine Berechtigung, eine bestimmte Aktion auf einem Objekt auszuführen. Ein Privileg bezieht sich hierbei immer auf ein Objekt.

Grundsätzlich unterscheidet die Fachliteratur folgende Zugriffskontrollmodelle: Discretionary Access Control, Mandatory Access Control und Rollenbasierte Zugriffskontrolle. Für einen Überblick werden diese im Folgenden beschrieben.

2.4.1 Discretionary Access Control (DAC)

Bei DAC werden zunächst Privilegien festgelegt, die explizit definieren, welche Subjekte welche Aktionen mit welchen Objekten ausführen dürfen. Zumeist werden Privilegien als Tripel der Form (Subjekt, Objekt, Aktion) dargestellt. Eine Anfrage eines Benutzers wird aufgrund dieser Regeln überprüft und entsprechend gewährt oder abgelehnt [dVFS07, CFMS94, FKC07]. Um die Privilegien gewähren zu können, beinhaltet DAC das Konzept des Eigentümers. Der Objekteigentümer hat hierbei die Vollmacht, anderen Subjekten Privilegien für dieses Objekt zu gewähren. Die wesentliche Charakteristik von DAC ist es deshalb, dass das Gewähren und Entziehen von Privilegien im Ermessen des individuellen Benutzers liegt. Ein Benutzer kann die Rechte seiner Objekte selbstständig ohne den Eingriff eines Systemadministrators verwalten.

Das erste DAC-Modell war die Zugriffsmatrix [GD71, HRU76, Lam74]. Seitdem wurden verschiedene weitere DAC-Modelle publiziert. Eine gute Übersicht findet sich in [CFMS94]. Eine wichtige Erweiterung des DAC-Modells war die Einführung von abstrakten Benutzern und Objekten. Hierdurch können hierarchisch organisierte Benutzergruppen und Objektegruppen definiert werden [dVFS07].

DAC kommt in den meisten kommerziellen DBMS zum Einsatz [EN07]. Insbesondere enthält der SQL Standard Operatoren zum Erteilen von Privilegien (Grant), Widerrufen von Privilegien (Revoke) und Weitergeben von Privilegien zur administrativen Delegation (With Grant Option) [EN07].

In DAC können positive und negative Privilegien vergeben werden. D.h., es kann ausgedrückt werden, was erlaubt ist (positiv) und was nicht erlaubt ist (negativ). Entsprechend sind alle Zugriffe, die nicht durch eine Regel abgedeckt sind, entweder nicht erlaubt (closed world assumption) oder zulässig (open world assumption). Typische Kombinationen sind positive Privilegien mit closed world assumption und negative Privilegien mit open world assumption.

DAC wird vor allem aufgrund seiner Flexibilität sehr oft eingesetzt. Aber DAC hat auch Schwächen. So ist es problematisch, dass nicht zwischen Benutzer und Subjekt unterschieden wird: Alle Programme, die im Auftrag des Benutzers laufen, tätigen ihre Anfragen mit allen Rechten des Benutzers. Dies macht das Zugriffskontrollsystem anfällig für bösartige Programme, insbesondere für Trojaner. Diese nutzen versteckte Funktionen, um die Autorisierung des Benutzers auszunutzen und damit Informationen unbemerkt an unberechtigte Dritte weiterzureichen. Zudem fehlt der DAC die Kontrolle über den Informationsfluss, sobald die Daten zugänglich gemacht wurden [dVFS07]. Auch ist die notwendige individuelle Festlegung der Privilegien insbesondere bei großer Anzahl von Nutzern oder Objekten sehr aufwändig.

2.4.1.1 Administrative Delegation

Die Verwaltung der Privilegien kann vom Objekteigentümer an andere Benutzer abgetreten werden. Dies wird als *administrative Delegation* bezeichnet, da der Objekteigentümer einen anderen Benutzer als Stellvertreter für die Administration der Privilegien an einem seiner Objekte ernennt. Es gibt verschiedene Ausgestaltungen dieser administrativen Delegation. Zumeist wird ein Benutzer ermächtigt, anderen Benutzern Rechte an einem bestimmten Objekt zu erteilen sowie dieses Privileg der administrativen Delegation wiederum an weitere Benutzer weiterzugeben. So entsteht ein sogenannter Delegationspfad (im Englischen Delegation Path) vom Objekteigentümer über die Benutzer, die das Privileg per administrativer Delegation weitergegeben haben, bis zum Besitzer des Privilegs. Das Entziehen von Privilegien wird ebenfalls durch administrative Delegation geregelt. Beim Widerruf eines Privilegs handelt es sich dann um eine Unterbrechung des Delegationspfades. Es gibt verschiedene Vorgehensweisen, wie mit einem unterbrochenem Delegationspfad umzugehen ist [HJPPW01]. Die übliche Vorgehensweise ist, alle auf dem entzogenen Privileg basierenden Privilegien ebenfalls zu entziehen. Dies entspricht dem kaskadierenden Entziehen der Privilegien (im Englischen Cascading Revoke).

2.4.2 Mandatory Access Control (MAC)

Um die soeben skizzierten Probleme von DAC zu lösen, unterscheidet die Mandatory Access Control (MAC) strikt zwischen Benutzer und Subjekt. Der Schutz bei MAC basiert auf der Klassifizierung von Subjekten und Objekten. Die Klassifizierung erfolgt nach Sicherheitsstufen und Kategorien. Sicherheitsstufen sind hierbei streng geordnet, während Kategorien ungeordnet sind. Sicherheitsstufen könnten z.B. „Streng Geheim“ (SG), „Geheim“(G) und „Vertraulich“(V); Kategorien „Wirtschaft“ und „Finanzwesen“ sein. Die Ordnung der Kategorien, auch Dominanz genannt, wird durch die Ordnungsbeziehung $\geq$ ausgedrückt. Die Ordnung der Sicherheitsstufen ist demnach SG $\geq$ G $\geq$ V. Die Kategorien werden dann den Sicherheitsstufen zugewiesen, z.B. SG (Finanzwesen, Wirtschaft) $\geq$ SG und G (Wirtschaft) $\geq$ G. Die Sicherheitsfreigabe des Benutzers muss immer die Sicherheitsfreigabe der Subjekte des Benutzers dominieren. Benutzer Bob, der Sicherheitsfreigabe G (Wirtschaft) besitzt, kann Sitzungen mit der Freigabe G (Wirtschaft), G oder V initiieren. Objekte werden ebenfalls den Sicherheitsstufen zugeordnet, je nachdem, wie sicherheitsrelevant die dort enthaltenen Daten sind [FKC07, dVFS07].

Im bekanntesten MAC-Modell von Bell-LaPadula [BL73], erfolgt die Zugriffskontrolle durch Auswertung von zwei Regeln:

Simple Security Property Kein Subjekt S darf ein Objekt O lesen, es sei denn, die Sicherheitsfreigabe von S dominiert die Sicherheitsstufe von O ($class(S) \geq class(O)$).

Star Property Ein Subjekt S darf kein Objekt O schreiben, es sei denn die Sicherheitsstufe des Objekts dominiert die Sicherheitsstufe des Subjekts ($class(O) \geq class(S)$).

Letztere Regel verhindert, dass ein Subjekt ein Objekt mit geringerer Sicherheitsstufe als der eigenen Sicherheitsfreigabe schreiben kann. Dadurch wird verhindert, dass Daten von einer höheren Sicherheitsstufe in ein Objekt einer geringeren Sicherheitsstufe geschrieben werden können, wodurch ihre Sicherheitsstufe herabgesetzt würde.

MAC-Modelle schützen die Daten zwar besser als DAC-Modelle (insbesondere vor Angriffen von Trojanern), aber sie sind nicht problemfrei. So werden nur Informationsflüsse

kontrolliert, die auf normalen, dafür vorgesehenen Wegen erfolgen. Es können aber andere Kanäle, die nicht zur Kommunikation gedacht sind, genutzt werden, um Informationen auszutauschen [dVFS07]. In der Praxis ist das Hauptproblem von MAC jedoch, dass es für viele Umgebungen zu starr und unflexibel ist. Insbesondere ist es nicht praktikabel, für jeden Benutzer Sicherheitsfreigaben und für jedes Datenobjekt Sicherheitsstufen festzulegen. Dies macht die Verwaltung der Zugriffskontrolle zu aufwändig.

Neuere Zugriffskontrollmodelle versuchen deshalb MAC und DAC miteinander zu kombinieren, um deren jeweiligen Vorteile zu nutzen und ihre Nachteile auszugleichen [CFMS94].

2.4.3 Rollenbasierte Zugriffskontrolle (RBAC)

Als dritte Alternative zur Zugriffskontrolle ist die rollenbasierten Zugriffskontrolle (RBAC) anzuführen [FKC07], deren Entwicklung vornehmlich von der Wirtschaft vorangetrieben wurde. Das grundlegende Konzept von RBAC ist es, den Zugriff auf Objekte von der Rolle des Benutzers in der Organisation abhängig zu machen. Dieses Prinzip wurde schon in den 70er Jahren in kommerziellen Computersystemen implementiert [FKC07]. Erst Mitte der 1990er Jahre integrierten Forscher die schon vorhandenen anwendungsspezifischen Ansätze in ein allgemeines RBAC-Modell [FK92, SCFY96]. Eines der Hauptziele von RBAC war die bessere Unterstützung der Verwaltung der Zugriffskontrolle in komplexen Umgebungen mit einer Vielzahl an Benutzern und Objekten [Osb07]. Dies war bei den bisherigen DAC- und MAC-Modellen mangelhaft.

Das ANSI-Standardmodell [NIS04] für RBAC basiert auf Benutzern, Rollen und Berechtigungen. Eine Rolle stellt hierbei ein „semantisches Konstrukt“ dar, für das Zugriffsregeln festgelegt werden [FKC07]. In RBAC werden dann den Berechtigungen Rollen zugeordnet. Benutzer werden zu Mitgliedern von Rollen gemacht und erben damit die Berechtigungen der Rollen. Die wichtigsten Zuordnungen sind deshalb:

Benutzerzuordnung Die Benutzerzuordnung ist die Zuweisung eines Benutzers zu Rollen, wobei ein Benutzer zu mehreren Rollen gehören kann und eine Rolle mehrere Benutzer haben kann.

Zuordnung der Berechtigungen Die Zuordnung der Berechtigung ist die Zuweisung von Berechtigungen zu einer Rolle, wobei wiederum eine Berechtigung zu mehreren Rollen gehören kann und eine Rolle mehrere Berechtigungen besitzen kann.

Rollen können in Hierarchien [SCFY96] angeordnet werden. Eine Rolle kann hierbei mehrere Rollen beinhalten und in mehreren Rollen enthalten sein. Berechtigungen bestehen aus der Beziehung zwischen Objekt und Aktion. Diese Beziehungen sind in Abbildung 2.2 dargestellt. Die Pfeile zwischen Benutzern und Rollen stellen hierbei die Benutzerzuordnungen dar, während die Zuordnung der Berechtigungen durch die Pfeile zwischen Rollen und Berechtigungen dargestellt ist. Eine Berechtigung besteht hierbei aus der Kombination von Aktion (hier read und write) und dem Objekt.

Für die Subjekte des Benutzers können verschiedene Rollen aktiviert werden. Durch diesen Mechanismus kann das Least Privilege Prinzip umgesetzt werden. Das Least Privilege Prinzip besagt, dass jedem Subjekt nur so viele Rechte gewährt werden sollen, wie dieses zur Ausführung seiner Aufgabe auch wirklich benötigt. Indem nur die Rollen aktiviert werden, die zur Bearbeitung der Aufgabe nötig sind, kann dies in RBAC umgesetzt werden.

Es ist wichtig zwischen Benutzergruppen, wie sie z.B. in Betriebssystemen bekannt sind, und Rollen zu unterscheiden. Bei Gruppen handelt es sich um eine Menge von Benutzern

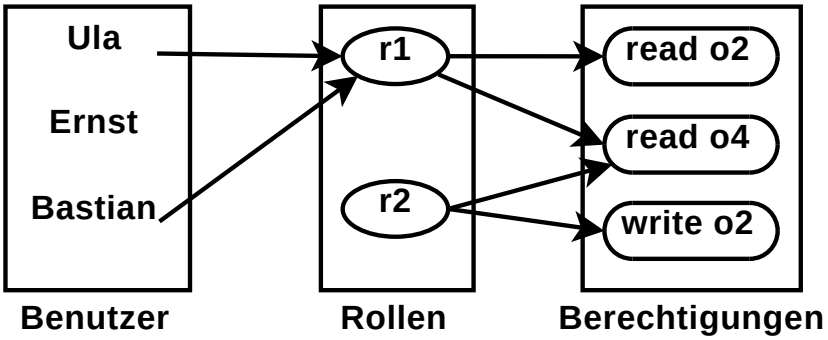


Abbildung 2.2: Beziehung zwischen Benutzer, Rollen und Berechtigungen in RBAC

und nicht um eine Menge von Berechtigungen. Zudem können Berechtigungen auch direkt Benutzern zugewiesen werden und nicht nur Gruppen. Rollen sind im Gegensatz zu den üblichen Gruppen hierarchisch organisiert.

Der RBAC-Standard enthält keine Vorgaben, wie die Administration von Rollen zu gestalten ist. Insbesondere, wer die Benutzerzuordnung und die Zuordnung der Berechtigungen vornimmt, ist nicht geregelt. Gleiches gilt für die Berechtigung Benutzer und Rollen anzulegen. Es gibt in der Forschung verschiedenste Ansätze diese Administration der Rollen zu regeln. Bei RBAC96 [SCFY96] gibt es spezielle administrative Rollen und Privilegien, die zum Verwalten der Zuordnungen berechtigen. Verschiedene Rollen haben hierbei einen unterschiedlichen administrativen Kontrollbereich, d.h. die von ihnen veränderbaren Zuordnungen sind festgelegt durch die ihnen zugeordneten administrativen Privilegien [FKC07]. Das ABRAC97 [SBM99] Modell baut auf RBAC96 auf und besteht aus den drei Komponenten Benutzerzuordnung (URA97), Zuordnung von Berechtigungen (PRA97) und Rollen zu Rollen Zuordnung (RAR97). ABRAC97 wurde mittlerweile von ABRAC02 [OS02] abgelöst. Daneben gibt es noch den Ansatz des Role Control Center [FCAG03] der die Verwaltung der Rollen in administrative Teilbereiche untergliedert, die wiederum hierarchisch organisiert sind.

Im Vergleich zu DAC und MAC ist RBAC allgemeiner gehalten wodurch es eine Vielzahl von Zugriffskontrollregeln unterstützt. Unter anderem können damit MAC [OSM00] und DAC [SM98] realisiert werden. Der wesentliche Vorteil von RBAC ist seine sehr gute Unterstützung in der Verwaltung von Privilegien und Benutzern in komplexen Umgebungen mit einer Vielzahl von Benutzern und Objekten. RBAC wird deshalb heute von vielen kommerziellen Systemen unterstützt, insbesondere von Datenbanken [FKC07, RSRS98]. Die Unterstützung für RBAC ist zudem im SQL-99 Standard enthalten.

2.4.4 Neuere Zugriffskontrollmodelle

Im Zuge der Service-orientierten Softwareentwicklung, wurden neue Zugriffskontrollmodelle entwickelt für verteilte, sich ständig ändernde Umgebungen. Hauptanliegen dieser Modelle ist es, dass der Serviceanbieter Bedingungen spezifizieren kann, die ein Benutzer erfüllen muss, damit er Zugriff auf diesen Service erhält.

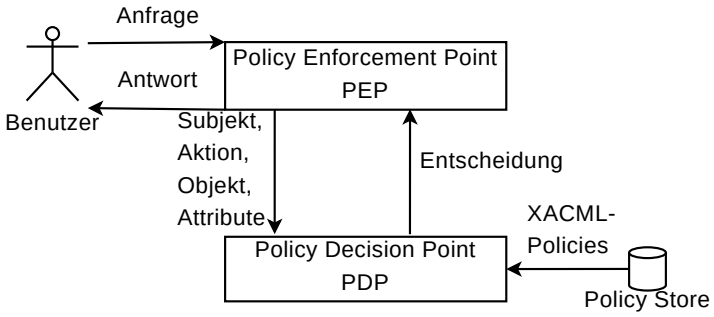


Abbildung 2.3: Der XACML-Evaluationprozess

2.4.4.1 eXtensible Access Control Markup Language (XACML)

XACML [Com05b] ist ein OASIS Standard für Zugriffskontrollregeln (in XACML auch Policy genannt). XACML ist XML-basiert und wurde insbesondere für den Austausch von Zugriffskontrollregeln entwickelt. Mit XACML ist es möglich, verschiedenste Policies miteinander zu kombinieren. Zudem lassen sich Privilegien gemäß Attributen von Benutzern und Objekten festlegen. Wichtig ist in verteilten Umgebungen zudem, dass XACML Policies von verschiedenen Stellen erstellt und durchgesetzt werden können und dass eine Policy auf andere Policies Bezug nehmen kann. Dies ist nur möglich, da XACML Policies unabhängig von der konkreten Implementierung der XACML Zugriffskontrolle [ADDS07] sind.

Eine XACML Policy besteht aus PolicySets, die andere PolicySets oder andere Policy Elemente zusammenfassen. Die wichtigsten Komponenten eines Policy Elements sind Target und Rule. Target spezifiziert den Anwendungsbereich der Policy und ein Rule Element bestimmt die zu überprüfenden Bedingungen. Jedes Rule Element enthält verschiedene Subject, Resource und Action Elemente. Das Subject Element spezifiziert hierbei die Attribute, die das Subjekt betreffen. Das Resource Element beschreibt die Bedingungen, die für die Ressource gelten. Das Action Element legt die Aktionen fest, die für ein Subjekt auf oder mit einer Ressource erlaubt sind. Der Eigentümer eines Objekts legt durch solch eine Regel fest, wem er erlaubt bestimmte Aktionen mit seinem Objekt durchzuführen. Ein Auszug aus einer XACML Policy ist in Abbildung 3.7 in Kapitel 3.3.1.1 zu sehen.

Im XACML-Standard ist der konkrete Ablauf der Zugriffskontrolle festgelegt. Wenn ein Benutzer auf ein Datenobjekt zugreifen möchte, werden zunächst alle XACML Policies gesammelt, ausgewertet und basierend auf dieser Auswertung eine Entscheidung gefällt (Zugriff erlaubt oder nicht). Dieser Ansatz ist sehr flexibel, da verschiedene Policies miteinander kombiniert werden und verschiedenste Prädikate während der Evaluation überprüft werden können. Der prinzipielle Ablauf einer solchen Evaluation ist in Abbildung 2.3 dargestellt. XACML unterscheidet hierbei zwischen dem Policy Enforcement Point (PEP) und dem Policy Decision Point (PDP). Der Benutzer interagiert nur mit dem PEP. Dieser nimmt die Anfrage des Benutzers auf, reichert diese um weitere Informationen wie Attribute des Benutzers oder des Datenobjekts an und leitet sie weiter an den PDP. Im PDP erfolgt dann die eigentliche Evaluation. Hierzu werden zunächst die nötigen Policies angefordert (in diesem Fall von einem Policy Store) und sobald alle vorhanden sind die Evaluation durchgeführt. Die Entscheidung der Evaluation wird dann an den PEP übermittelt der entsprechend die Anfrage des Benutzers akzeptiert oder zurückweist.

XACML unterstützt RBAC [NIS04] durch die RBAC-Spezifikation für XACML [Com05a].

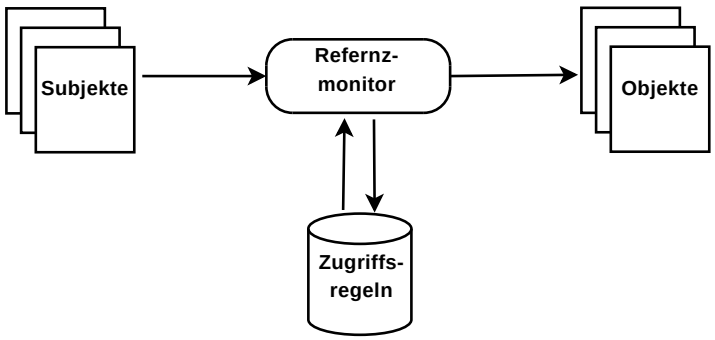


Abbildung 2.4: Referenzmonitor

Benutzer und Rollen werden dabei auf XACML Subjects, Objekte auf XACML Resources, und Aktionen auf XACML Actions abgebildet. Die Benutzerzuordnung und die Zuordnung der Berechtigungen wird über zusätzliche Policies geregelt.

Die Vergabe von Privilegien an Benutzer ist im XACML-Standard nicht enthalten. Es ist nicht festgelegt, wer Policies ändern, löschen oder erstellen darf. Rissanen und Firozabadi [RF04] schlagen deshalb vor, diese Administration einer Policy wieder in einer XACML Policy festzulegen. Diese Idee wurde als Administrative Policy in den momentanen Working Draft für XACML Version 3.0 aufgenommen [R$^+$07].

2.4.4.2 Akenti

In Akenti [TEM03, Ake] werden X.509 Zertifikate [Tel05] neben der Authentifizierung auch noch zur Autorisierung von Benutzern in einem verteilten System benutzt. Hierzu werden Zugriffsregeln in vom Objekteigentümer signierte Zertifikate eingefügt. Akenti Zugriffsregeln werden in XML ausgedrückt und verteilt im System gespeichert. Im Akenti Modell entsprechen die Objekte Ressourcen auf die über ein Resource Gateway zugegriffen wird. Das Resource Gateway stellt hierbei den Ort dar, an dem die Zugriffskontrolle erfolgt. Der Eigentümer einer Ressource spezifiziert die Zugriffsbedingungen in der Form von signierten Zertifikaten. Zur Überprüfung der Rechtmäßigkeit eines Zugriffs sucht Akenti zunächst die relevanten Zertifikate und überprüft anhand derer dann die Rechte des Benutzers für diese Ressource.

2.5 Mechanismen zur Durchsetzung der Zugriffskontrolle

Alle vorgestellten Zugriffskontrollmodelle spezifizieren nur die Zugriffsregeln. Durch wen, wo und wie diese Regeln ausgewertet werden, bleibt dem konkreten Zugriffskontrollmechanismus überlassen. Akenti und XACML legen zwar fest, wie die Auswertung ihrer Policies erfolgt, jedoch auch nicht wo und durch wen. Die Durchsetzung und die Auswertung der Zugriffskontrollregeln übernimmt der Referenzmonitor. Dies ist schematisch in Abbildung 2.4 dargestellt. Hier wird deutlich, dass der Referenzmonitor die Anfragen der Subjekte zunächst prüft, indem er die Zugriffsregeln auswertet. Nur bei erfolgreicher Prüfung erfolgt die Weiterleitung der Anfrage an die Objekte. In der Literatur finden sich drei Möglichkeiten, wer für die Durchsetzung der Zugriffskontrollregeln verantwortlich ist: serverseitige Durchsetzung, clientseitige Durchsetzung und gruppenbasierte Durchsetzung.

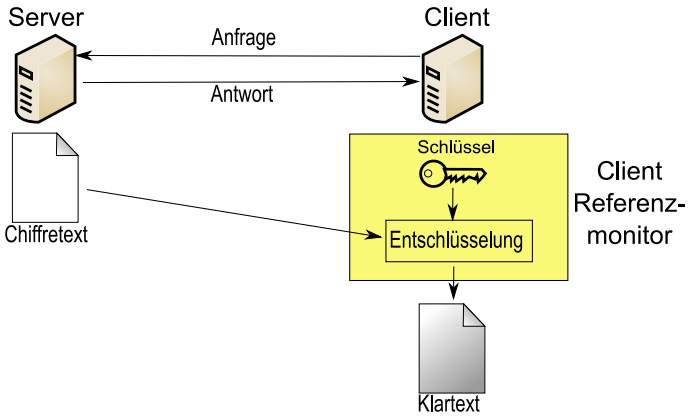


Abbildung 2.5: Clientseitige Durchsetzung

2.5.1 Serverseitige Durchsetzung

Bei der serverseitigen Durchsetzung handelt es sich um die klassische Lösung, die von den meisten Zugriffskontrollmodellen implizit angenommen wird. Durchsetzung bezeichnet die Auswertung der Zugriffskontrollregeln und ihre konstante Einhaltung. Der Referenzmonitor des Datenanbieters (Server) ist hierbei für die Durchsetzung der Zugriffsregeln verantwortlich. Daten werden nur dann vom Server ausgeliefert, wenn der Anfragende hierzu autorisiert ist.

Diese serverseitige Durchsetzung unterstützt alle in Kapitel 2.4 vorgestellten Zugriffskontrollmodelle vollumfänglich. Probleme können jedoch im Zusammenhang mit Replikation auftreten. Der Server, der die Replikate speichert, muss für die Replikate auch die Durchsetzung der Zugriffskontrolle durchführen. Hierzu ist ein entsprechendes Vertrauen in den Replikatserver erforderlich. Dieses kann, wie von [MS03] dargestellt, in bestimmten Umgebungen fehlen, wie z.B. bei „Database as a Service“. Für diese Umgebungen ist die serverseitige Durchsetzung deshalb nicht geeignet.

2.5.2 Clientseitige Durchsetzung

Bei der clientseitigen Durchsetzung erfolgt ein Teil der Durchsetzung beim Anfragenden. Ein Teil des Referenzmonitors befindet sich deshalb auf dem Client, der die Anfrage gestellt hat. Das Grundprinzip ist hier, dass die Daten zunächst von Objekteigentümer und dessen Teil des Referenzmonitors vorverarbeitet werden. Im Allgemeinen erfolgt dort eine Verschlüsselung der Daten gemäß der Auswertung der Zugriffsregeln. Die Schlüssel, die zur Entschlüsselung der Datenobjekte nötig sind, werden dann über einen sicheren Kommunikationskanal an die Benutzer entsprechend ihrer Privilegien verteilt (z.B. [MS03], [BCF+04] oder [BNP04]). Dies ist in Abbildung 2.5 dargestellt. Eine Anfrage bei einem solchen Zugriffsmechanismus läuft wie folgt ab.

Zunächst stellt der Client die Anfrage an den Server. Im Gegensatz zu der serverseitigen Zugriffskontrolle wird der Server das angeforderte verschlüsselte Datenobjekt immer an den Client ausliefern. Wurde das verschlüsselte Datenobjekt übermittelt, übernimmt nun der Referenzmonitor auf dem Client die Entschlüsselung. Hierzu benötigt er den entsprechenden Schlüssel, der zuvor über geeignete sichere Wege den Clients zugänglich gemacht wurde. Nur falls der Client über den entsprechenden Schlüssel verfügt, kann der Referenzmonitor das Objekt entschlüsseln und somit dem Client Zugriff auf das Datenobjekt gewähren. Beispiele

für solche Zugriffskontrollen sind [HILM02] und [BCF01].

Die Vorteile der clientseitigen Durchsetzung sind vielfältig. So unterstützt sie die Replikation auch in Umgebungen mit geringem Vertrauen zwischen den Teilnehmern. Zudem verteilt sie die Arbeitslast der Zugriffskontrolle auf die Teilnehmer. Allerdings hat die clientseitige Durchsetzung auch einige Nachteile. Ein Entziehen von Privilegien bedingt eine erneute Verschlüsselung der Daten. Grund hierfür ist, dass die Schlüssel zur Entschlüsselung der Daten dem Client ausgehändigt werden. Soll einem Client ein Privileg entzogen werden, muss entsprechend ein neuer Schlüssel generiert werden, die Daten mit dem neuen Schlüssel verschlüsselt und der neue Schlüssel an die nach wie vor berechtigten Benutzer ausgehändigt werden (Revocation Problem). Ein weiteres Problem ist, dass Privilegien immer an Schlüssel gebunden sind. Dies schränkt die Zugriffskontrollmodelle ein, die damit unterstützt werden können.

2.5.3 Gruppenbasierte Durchsetzung

Bei der gruppenbasierten Zugriffskontrolle werden die Daten auf dem Server ebenfalls verschlüsselt abgelegt. Der Referenzmonitor wird allerdings jetzt weiter aufgegliedert. So befindet sich der Referenzmonitor auch auf einer Gruppe zuvor ausgewählter Stellvertreter, im Folgenden auch Delegates genannt. Diese übernehmen stellvertretend für den Dateneigentümer die Durchsetzung der Zugriffsregeln, indem sie einen Teil der Ver- und Entschlüsselung der Daten übernehmen. Die Stellvertreter führen hierbei eine t aus n Schwellenentschlüsselung (siehe Kapitel 2.3.3.2) durch. Nur falls t Stellvertreter das Datenobjekt partiell entschlüsselt haben, kann der Anfragende daraus das vollständig entschlüsselte Datenobjekt generieren. Durch diesen Mechanismus reduziert das Verfahren das nötige Vertrauen in den einzelnen Stellvertreter und ist resistent gegenüber Ausfällen einzelner Stellvertreter. Auch bösartige Stellvertreter können durch das Schwellensystem keinen Schaden anrichten, solange nicht t der n Stellvertreter bösartig sind und alle miteinander kooperieren. Da die Schwellenentschlüsselung keinerlei Informationen über den Schlüssel preisgibt, der zur Entschlüsselung erforderlich ist, tritt das Revocation Problem nicht auf. Zudem sind die Privilegien nicht an Schlüssel gebunden, weshalb beliebige Zugriffskontrollregeln unterstützt werden können. Jeder Stellvertreter kann als serverseitige Referenzmonitor angesehen werden, der aufgrund der ihm vorliegenden Information entscheidet, ob er den Zugriff gewährt oder nicht. Dementsprechend erfolgt dann die partielle Entschlüsselung und Lieferung des Datenobjekts an den Client oder nicht. Dieses Vorgehen stellt auch gleichzeitig den Nachteil dieses Verfahrens da denn es verursacht hohe Kommunikationskosten. Die Daten müssen von mindestens t Stellvertretern teilweise entschlüsselt und an den Client ausgeliefert werden.

Beispielanwendungen für diese Durchsetzungsart sind Authentifizierungsdienste [Gon93] und replizierte Dienste [RB94]. Eine abgewandelte Form mit der direkten Aufteilung des Datenobjekts auf die Stellvertreter mit einem t von n Schwellenschema wurde von Rabin [Rab89] vorgestellt und von Krawcyk [Kra93] erweitert.

Des Weiteren gibt es eine verbreitete Abwandlung der gruppenbasierten Durchsetzung, in dem die Stellvertreter lediglich der Verifizierung des gelieferten Datenobjektes dienen oder aber die Stellvertreter nur den Schlüssel zur Entschlüsselung des Datenobjektes liefern. In diesem Fall kommen häufig Schwellensignaturverfahren zum Einsatz (siehe Kapitel 2.3.3.3). Dies wird z.B. bei Herlihy und Tygar [HT88] oder Naor und Wool [NW98] für verteilte Datenbanken vorgestellt.

2.6 Föderierte Datenbanksysteme

Föderierte Datenbanken [SL90] gelten als Basis der in dieser Arbeit behandelten P2P-Datenbanken. Sie behandeln die Integration heterogener Datenquellen und integrieren hierbei auch die Zugriffskontrollmodelle und die Benutzerverwaltung. Eine Übersicht über die Gemeinsamkeiten und Unterschiede zwischen P2P und föderierten Datenbanken findet sich in [BCOS08].

2.6.1 Definition föderierte Datenbanken

Ein föderiertes Datenbanksystem besteht aus mehreren eigenständigen Datenbanksystemen, die zu einem gemeinsamen Datenbanksystem (dem föderierten) integriert werden. Die zu integrierenden Datenbanken werden hierbei zu Komponentendatenbanken. Ihre interne Organisation und Struktur bleibt unverändert. Diese Komponentendatenbanken können anschließend lose oder eng gekoppelt werden [Con97].

Eng gekoppelt Bei einem eng gekoppelten föderierten Datenbanksystem existiert ein Föderierungsdienst, der globalen Anwendungen den gesamten Datenbestand der Föderation zugänglich macht und die Daten der Komponentendatenbanken in das globale Schema der föderierten Datenbank entsprechend integriert.

Lose gekoppelt Bei lose gekoppelten föderierten Datenbanken kommunizieren die Komponentendatenbanken direkt miteinander. Jede Komponentendatenbank baut ihre Föderation dadurch quasi selbst. Es gibt keine übergeordnete Instanz (Föderierungsdienst), die das globale Schema verwaltet.

2.6.2 Zugriffskontrolle in lose gekoppelten föderierten Datenbanken

Die Zugriffskontrolle in lose gekoppelten föderierten Datenbanken besteht aus zwei Komponenten: Im *Exportschema* werden die Datenelemente angegeben, die der Föderation zur Verfügung gestellt werden. In der *Zugangsliste* sind für jedes Element des Exportschemas [HM85], lediglich die anderen Datenbanken der Föderation und nicht deren Benutzer enthalten. So ist es selbst bei einer Datenbank mit fehlerhafter Benutzerauthentifizierung nicht möglich, die Zugriffskontrolle der gesamten Föderation zu umgehen.

2.6.3 Zugriffskontrolle in eng gekoppelten föderierten Datenbanken

Die Zugriffskontrolle besteht in eng gekoppelten föderierten Datenbanken aus der Zugriffskontrolle auf den lokalen Datenbanken der Teilnehmer und einer Zugriffskontrolle auf dem Föderierungsdienst, der als zentrale Koordinator fungiert. Der Föderierungsdienst enthält insbesondere Abbildungen zwischen den Teilnehmern, um die Zugriffskontrollen auf den lokalen Datenbanken zu koordinieren. Da die Komponentendatenbanken ihre eigenen lokalen Zugriffskontrollsysteme besitzen, muss eine Abbildung der globalen Benutzer auf lokale Benutzer der Komponentendatenbanken erfolgen. Zudem haben die verschiedenen Komponentendatenbanken unterschiedliche Zugriffskontrollmodelle [Jon98, dVS96, CdVF97, dVS97].

2.7 Peer-to-Peer Systeme

Peer-to-Peer (P2P) als Netzwerkarchitektur ist nicht neu. So gab es erste Untersuchungen dieser Architektur schon Anfang der 90er Jahre (siehe z.B. [Sim91]). Es gibt unterschiedliche Definitionen von Peer-to-Peer. Für die vorliegende Arbeit soll die Definition von [SvHSS06] gelten: Der Begriff „Peer-to-Peer“ beschreibt ein System mit einer dezentralen Architektur, die es dem individuellen Peer erlaubt, Ressourcen anzubieten und zu nutzen ohne zentrale Kontrolle (eigene Übersetzung). Da in der vorliegenden Arbeit Systeme zum Informationsaustausch betrachtet werden, stellen Ressourcen in diesem Zusammenhang immer Daten dar. Aus der Definition geht hervor, dass Peers ein sehr hohes Maß an Autonomie innerhalb des Netzwerks besitzen. Die Definition sagt zudem bewusst nichts über die Charakteristik der einzelnen Peers aus. Häufig werden jedoch mit P2P auch Eigenschaften verbunden, die von der Charakteristik der Peers abhängen. Hierzu gehört, dass die Verknüpfungen zwischen den Teilnehmern sehr variabel sind. Des Weiteren können die Teilnehmer nur temporär im Netzwerk verfügbar sein, was zu einer hohen Dynamik des Systems beiträgt. P2P-Systeme, die diese zusätzlichen Eigenschaften besitzen, werden deshalb im Weiteren als *dynamische* P2P-Systeme bezeichnet.

Nach [Sch01] können P2P-Netzwerke weiter in *hybride* und *reine* P2P-Netzwerke aufgeteilt werden. Bei reinen P2P-Netzwerken gilt neben der allgemeinen Definition zu P2P, das jeder beliebige Netzwerkteilnehmer entfernt werden kann, ohne dass dadurch der vom Netzwerk angebotene Dienst ausfällt. Entsprechend gilt für ein hybrides P2P-Netzwerk, das eine zentrale Komponenten für Teile des vom Netzwerk angebotenen Dienstes nötig ist. Zwischen reinen und hybriden P2P-Netzwerken gibt es verschiedene Zwischenstufen.

Die Motivation, P2P-Systeme zu entwickeln, erklärt sich aus den vielen positiven Eigenschaften ihrer Netzwerkstruktur [ATS04] [MKRL+03]. P2P-Systeme sind:

- hoch skalierbar, da sie auf keine zentrale Komponente vertrauen
- selbstorganisierend; dadurch vermeiden sie Administrationskosten für eine zentrale Fehlererkennung und -behebung.
- fehlertolerant; dadurch bleiben sie selbst in Umgebungen, in denen die Knoten nur für kurze Zeit dem Netzwerk beitreten und/oder Netzwerk- und Computerfehler häufig auftreten, funktionstüchtig

Das Speichern und Abfragen von Daten in einem P2P-Netzwerk ist immer abhängig von den Peers (Knoten) im Netzwerk und den Verbindungen (Kanten) zwischen ihnen. Dieses Netzwerk ist vom zugrundeliegenden physischen Kommunikationsnetzwerk unabhängig und wird deshalb als Overlay-Netzwerk bezeichnet. Die Struktur, die Organisation (von zentral bis dezentral) und die Topologie dieses Overlay-Netzwerks beeinflussen die Funktionsweise des Netzwerks erheblich. Insbesondere die auf dem Overlay-Netzwerk aufsetzenden Routing- und Suchmechanismen sind kritisch, da sie fast alle grundlegenden Eigenschaften des Systems (wie Fehlertoleranz, Leistungsfähigkeit, Skalierbarkeit) maßgeblich bestimmen [ATS04].

In den folgenden Kapiteln werden die bekanntesten Routing- und Suchmechanismen vorgestellt. Da diese von der Struktur des Overlay-Netzwerks abhängig sind, werden sie entsprechend dieser Struktur kategorisiert. Ausschlaggebend ist hierbei, ob das Overlay-Netzwerk zufällig erzeugt wird (*unstrukturiertes* P2P-Netzwerk) oder ob beim Erzeugen gewisse Regeln über den Aufbau bestimmen (*strukturiertes* P2P-Netzwerk).

2.7.1 Unstrukturierte P2P-Netze

Bei unstrukturierten P2P-Netzen wird das Overlay-Netzwerk zufällig erzeugt und die Daten zufällig über das Netzwerk verteilt. Deshalb müssen Daten im Netz gesucht werden. Die dabei verwendeten Suchmechanismen sind vielfältig. Beispiele sind:

Flooding Flooding bezeichnet das rekursive Weiterleiten von Nachrichten an *alle* Knoten, die dem Knoten bekannt sind, bis die Daten gefunden wurden. Als Abbruchkriterium für die Weiterleitung wird die maximale Lebenszeit (im Englischen „Time to Live“) oder die maximale Anzahl an Weiterleitungen (Hops) für jede Nachricht festgelegt.

Random Walks Bei Random Walks werden die Nachrichten rekursiv an *zufällig ausgewählte* Nachbarn weitergeleitet, bis die Daten gefunden wurden. [LCC+02]

Routing Indexe Das Weiterleiten der Nachrichten erfolgt gemäß eines lokalen Routing Indexes, der die Daten der Nachbarknoten enthält [YGM02].

Unstrukturierte P2P-Netzwerke bleiben selbst bei einer hohen Fluktuation der Teilnehmer funktionstüchtig. Allerdings haben vor allem die einfachen Routingmechanismen wie Flooding Skalierbarkeitsprobleme. Dies war auch der Grund für die Einführung ausgefeilterer Routingtechniken wie Random Walks durch [LCC+02] oder Lokaler Routing Indexe [YGM02]. Eine gute Übersicht über weitere Techniken zur Effizienzsteigerung findet sich in der Zusammenstellung von Androutsellis-Theotokis und Spinellis [ATS04].

2.7.1.1 P2P-Netze mit Super Peers

Eine Besonderheit unstrukturierter Netze stellen Supernode P2P-Netze dar mit zwei Klassen von Knoten: den normalen Knoten und den Super Knoten. Diese Architekturvariante liegt zwischen reinen und hybriden P2P-Systemen, da es sich um eine partiell zentralisierte Architektur handelt. Die Einteilung der Knoten in diese Klassen erfolgt aufgrund der Ressourcen der Knoten. Super Peers indexieren Dateien der normalen Knoten, die mit ihnen verbunden sind. Dadurch können die Super Peers Suchanfragen stellvertretend für die verbundenen normalen Knoten beantworten. Vorteil dieser Architekturvariante des unstrukturierten P2P-Netzes ist die effizientere und schnellere Suche und die daraus resultierende bessere Ausnutzung der zur Verfügung stehenden Ressourcen [YGM03].

2.7.2 Strukturierte P2P-Netze

Bei strukturierten P2P-Netzwerken ist die Anordnung der Knoten im Overlay-Netzwerk vorgegeben. Ebenso ist genau festgelegt, wo (auf welchem Knoten) ein bestimmtes Datenobjekt im Overlay-Netzwerk gespeichert wird. Diese Netzwerke ermöglichen deshalb eine Abbildung von darin gespeicherten Datenobjekten (z.B. anhand des Dateinamens) auf deren Speicherort (z.B. IP-Adresse des Knotens). Routingtabellen auf den Peers sorgen hierbei für ein effizientes Routing von Nachrichten.

Aufgrund dieser exakten Abbildung von Inhalt auf Speicherort bilden strukturierte P2P-Netzwerke eine sehr leistungsfähige und skalierbare Lösung für exakte Suchen, d.h. für Suchen, bei denen z.B. der exakte Dateiname bekannt ist. In diesem Fall beschränkt sich die Anzahl der nötigen Routingschritte bei allen Ansätzen (bis auf CAN [RFH+01]) auf $\mathcal{O}(\log N)$, wobei N die Netzwerkgröße darstellt.

Strukturierte Netzwerke bieten hingegen kaum Unterstützung für Bereichsanfragen, wie z.B. Suchen nach Dateinamen, die mit „A“ beginnen. Ein weiterer Nachteil strukturierter P2P-Netzwerke ist der Aufwand, den die Verwaltung dieser Struktur erzeugt. Dies gilt insbesondere bei hoher Fluktuation der Netzwerkteilnehmer [ATS04].

Im Folgenden werden die bekanntesten Routingsysteme für strukturierte P2P-Netze dargestellt. Hierbei findet Chord [SMK+01, SMLN+03] resultierend aus seiner Bedeutung für die vorliegende Arbeit besondere Beachtung.

2.7.2.1 Strukturierte Overlay-Netzwerke

Chord Das Massachusetts Institute of Technology (MIT) entwickelte im Projekt Chord [SMK+01, SMLN+03, DBK+01, Cho] eine Lösung für das effiziente Finden des Peers, auf welchem ein bestimmter Datensatz gespeichert ist. Die Aufgabe des Chord Protokolls besteht deshalb darin, die Zuordnung vom Schlüssel zu seinem Speicherort (Knoten) vorzunehmen. Diese Zuteilung erfolgt durch konsistentes Hashing [KLL+97], das eine gleichmäßige Lastenverteilung auf die Knoten gewährleistet. Chord ist zudem ein verlässliches Protokoll, d.h., eine Antwort ist garantiert. Das Chord Protokoll legt fest, wie der Ort eines Schlüssels gefunden werden kann und was bei Fehlern zu tun ist. Die Hashfunktion weist jedem Schlüssel und jedem Knoten eine Referenznummer zu, ChordID genannt. Die Knotenreferenznummer wird durch das Hashing der IP-Adresse des Knotens erzeugt, die Schlüsselreferenznummer durch das Hashing des Schlüssels. Die Referenznummern sind in einem Referenzring angeordnet. Der Schlüssel s wird nun dem Knoten zugewiesen. Für dessen Referenznummer r gilt, dass $r \leq s$ und dass es keinen Knoten r' gibt für den $r < r' < s$ gilt. Dieser Knoten wird auch *Successor* Knoten genannt. Tritt ein neuer Knoten dem Netz bei oder verlässt ein Knoten dasselbe, müssen die darauf abgelegten Schlüssel an die nachfolgenden Knoten abgegeben bzw. von diesen neu verteilt werden. Jeder Knoten braucht deshalb nur seinen Successor-Knoten zu kennen. Um das Routing effizienter zu gestalten, besitzt jeder Knoten eine *Fingertabelle*, in der ein Teil der Knoten des Netzwerks aufgeführt sind. Die Einträge besitzen bei Chord jeweils eine ChordID und den Successor-Knoten zu dieser ChordID. Das Routing der Nachrichten in Chord erfolgt immer im Uhrzeigersinn im Ring. Die Fingertabellen enthalten entsprechend nur ChordIDs, die ihrer eigenen ChordID in der Ringstruktur folgen. Die in Abbildung 2.6 in Anlehnung [SMK+01] dargestellt Chord Ringstruktur hat einen Schlüsselraum von $[0, 7]$. Dargestellt ist ein Chord Netzwerk mit drei Knoten mit den ChordIDs null, eins und drei mit ihren Fingertabellen. Zudem sind darin drei Schlüssel mit den ChordIDs eins, zwei und sechs gespeichert. Jeder Eintrag in der Fingertabelle besteht aus der ChordID und dem Successor für diese ChordID. Die Schlüssel sind bei ihrem jeweiligen Successor-Knoten gespeichert. Der Schlüssel sechs ist deshalb bei Knoten null gespeichert und Schlüssel zwei bei Knoten drei.

Die Dynamik des Systems kann mit wenig Aufwand verwaltet werden. Um allerdings Anfragen auch während der Umstrukturierung des Netzes korrekt bearbeiten zu können, ist ein eigenes Serialisierungsprotokoll nötig. Eine genauere Untersuchung der Leistungsfähigkeit von Chord befindet sich in [LNBK02]. Ein Ansatz zur Reduzierung des Kommunikationsaufwands für Chord wurde von [EAABH03] vorgestellt.

P-Grid Um eine selbstoptimierende Strukturierung des Netzwerks zu erlangen, wurde an der EPFL Lausanne das P-Grid Protokoll entwickelt [APHS02, ACMD+03a, P-G]. Durch P-Grid ist eine Strukturierung nach Sucheffizienz und/oder Ressourcenauslastung möglich, was der Lastverteilung zugute kommt. Grundlage bildet ein virtueller, verteilter Suchbaum. Jeder

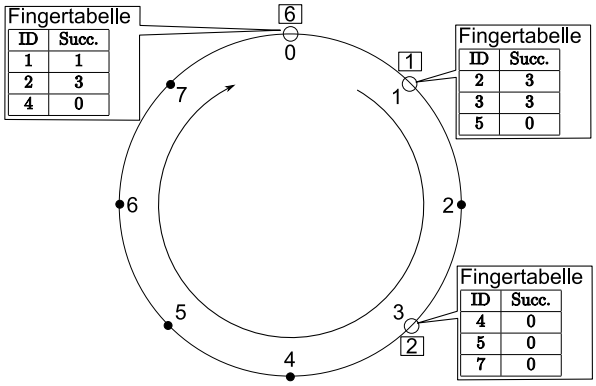


Abbildung 2.6: Die Ringstruktur von Chord (in Anlehnung an [SMK+01])

Knoten erhält hierbei einen Teil des gesamten Baumes und ist durch seinen Pfad im Baum adressierbar. Dementsprechend repräsentiert dieser Pfad auch den Teil des Baumes, für den der Knoten zuständig ist.

Durch autonome lokale Entscheidungen der Peers, wird eine gleichmäßige Verteilung der Last erreicht z.B. im Bezug auf den Speicherplatz. Daraus resultiert allerdings ein Zuwachs der Suchkosten, da der Baum nun nach dem Speicherplatz aufgebaut ist. Eine weitere Möglichkeit die Leistungsfähigkeit zu verbessern ist die Replikation der Daten. Die Entscheidung, ob und wo Replikate erzeugt werden, erfolgt wiederum lokal bei den betroffenen Peers. Weitere Aufgabe des Protokolls ist das Erkennen und Verwalten der dynamischen Adressen der Peers [ACMD+03b]. Unterstützung für Aktualisierungen der Daten wurden ebenfalls integriert [DHA03].

P-Tree Bei P-Tree [CLM+04, CLGS04] wird eine Indexstruktur aufgebaut, die eine verlässliche Verbindung zwischen den Peers herstellt. Entsprechend ist auch bei Netzwerkfehlern oder Ausfällen einzelner Peers ist die Erreichbarkeit des vollständigen Indexes gewährleistet. Zudem wird durch die Indexstruktur eine gleichmäßige Lastverteilung ermöglicht.

CAN CAN (Content Addressable Network) [RFH+01] baut eine Hashtabelle auf, die auf die Teilnehmer verteilt wird. Jeder Teilnehmer erhält dabei einen definierten Teil der gesamten Hashtabelle zugewiesen. Des Weiteren besitzt jeder Teilnehmer Informationen über benachbarte Teile der Hashtabelle, insbesondere welcher Teilnehmer diese Teile gespeichert hat. Anhand dieser Informationen findet dann das Routing der Nachrichten statt. Die Abbildung der Schlüsselwerte auf die Hashtabelle erfolgt über ein mehrdimensionales kartesisches Koordinatensystem. Datenobjekte werden hierbei auf einen Punkt in diesem Koordinatenraum abgebildet. Der Teil der Hashtabelle, für die ein Teilnehmer verantwortlich ist, entspricht einem Bereich dieses Koordinatenraums. Der Peer, in dessen Segment der Abbildungspunkt des Datenobjektes liegt, ist für dessen Speicherung verantwortlich.

Pastry Pastry [RD01, CDHR03, Freb] weist den Knoten im Netzwerk eindeutige Identifikationsnummern (Knoten IDs) zu und reiht sie gemäß diesen IDs in einen zirkulären ID Raum ein. Für Datenobjekte werden ebenfalls IDs berechnet. Zudem wird festgelegt, dass Datenobjekte immer auf dem Knoten gespeichert werden, dessen ID der Datenobjekt-ID nu-

merisch am nächsten ist. Beim Routing wird die Nachricht dann immer an den Knoten mit der numerisch nächsten ID weitergeleitet. Die Menge der von Pastry gespeicherten Routinginformationen zu anderen Peers sinkt mit zunehmender Entfernung der Knoten. Im Gegensatz z.B. zu Chord speichert Pastry hierbei Informationen von Vorgängern (Knoten mit kleinerer ID) und Nachfolgern (Knoten mit größerer ID). Das Routing von Nachrichten ist durch diesen Mechanismen sehr effizient.

P-Ring P-Ring [CLM+07] unterstützt Bereichsabfragen. Durch das von den anderen Systemen verwendete Hashing wird die Ordnung der Schlüsselwerte normalerweise zerstört. P-Ring verzichtet deshalb darauf und weist den Peers die Objekte gemäß ihres Schlüssels direkt zu. Da Objekte mit aufeinanderfolgenden Schlüsseln entsprechend auch auf denselben oder benachbarten Peers gespeichert sind, werden hierdurch Bereichsabfragen möglich. Allerdings wird durch die potentiell asymmetrische Schlüsselverteilung (und die fehlende Hashfunktion) eine gleichmäßig Verteilung der Knoten verhindert. In P-Ring werden deshalb die Schlüsselbereiche, für die ein Peer zuständig ist, dynamisch angepasst, um so eine gleichmäßige Verteilung der Objekte zu erreichen.

BATON BATON [JOV05] und die Erweiterung BATON* [JOT+06] basieren auf einem ausbalancierten Baum. Jeder Netzwerkknoten verwaltet genau einen Knoten dieses Baums. Natürlich muss jeder Peer zur effizienten Verwaltung des Baumknotens Verbindungen zu den Nachbar-, Kind- und Elternknoten besitzen. Jedem Knoten ist ein Wertebereich zugeordnet, für dessen Speicherung er zuständig ist. Durch das dynamische Verwalten dieser Wertebereiche und den Verzicht auf das Hashing der Schlüsselwerte werden Bereichsabfragen unterstützt. Während BATON einen binären Baum verwaltet, kann BATON* auch Bäume mit mehr als zwei Kindern unterstützen. Das Routing der Nachrichten erfolgt durch die Baumstruktur sehr effizient. Allerdings erzeugt die Wartung dieser Struktur auch erheblichen Aufwand, insbesondere wenn Knoten das Netzwerk verlassen oder hinzukommen.

2.7.2.2 Verteilte Hashtabelle (DHT)

Eine verteilte Hashtabelle, aufgrund des englischen Fachbegriffs „Distributed Hashtable“ auch kurz DHT genannt, stellt die gleiche Funktionalität wie eine normale Hashtabelle zur Verfügung [DZD+03]. In ihr werden Abbildungen von Schlüsseln zu ihren Werten abgespeichert. Besonderes Merkmal der DHT ist jedoch, dass die Hashtabelle nicht als Ganzes gespeichert, sondern auf die Teilnehmer des Netzwerks aufgeteilt wird. DHT ist deshalb ein Sammelbegriff bestimmter erweiterter Funktionalitäten der zuvor vorgestellten strukturierten P2P-Netzwerken. Eine Hashtabelle unterstützt entsprechend die Operationen zum Erlangen eines Datenobjektes *get(key)*, mit dem entsprechenden Schlüssel des gesuchten Datenobjekts als Funktionsparameter und dem Wert als Rückgabeparameter, der diesem Schlüssel zugewiesen wurde. Die entsprechende Funktion zum Speichern eines Schlüssel- (key) Wert- (value) Paares ist *put(key,value)*. Die strukturierten Netzwerke übernehmen hierbei die Abbildung des Schlüssels auf die Netzwerkstruktur, d.h. sie legen fest, auf welchem Peer das jeweilige Schlüssel-Wert-Paar gespeichert wird. Dabek u.a. [DZD+03] haben hierfür das Common API vorgestellt, das die zur Verfügung gestellten Funktionen der strukturieren Netzwerke zusammenfasst und damit eine einheitliche Schnittstelle für die darauf aufbauenden DHTs etabliert. Bei einer DHT können Werte Datenobjekte beliebiger Art sein. Der Vollständigkeit

halber gibt es noch eine Funktion zum Löschen eines Schlüssels und damit des Datenobjektes *remove(key)*.

Es gibt unzählige Anwendungen, die auf DHTs aufbauen und entsprechend erweiterte Funktionalitäten bereitstellen. Beispielhaft sei hier das Chord-basierte CFS bzw. DHash [DKK+01] und das Pastry basierte PAST [DR01] genannt.

Im Folgenden werden DHT und strukturierte Netzwerke synonym verwendet, da alle hier vorgestellten strukturierten Netzwerke lediglich dem Ziel des Aufbaus einer DHT dienen. Andere Anwendungen der strukturierten Netzwerke, wie dezentrale Verzeichnisdienste oder skalierbare Gruppenkommunikation, werden in dieser Arbeit nicht betrachtet.

2.8 P2P-Datenbanken

Es gibt eine Vielzahl von Forschungsgruppen, die sich mit Datenbanken im P2P-Umfeld beschäftigen. Hierbei wird zumeist die Peer-to-Peer Netzwerkstruktur als eine neue Herausforderung für Datenintegrationssysteme wie föderierte Datenbanken betrachtet. Das Ziel der Datenintegration ist ein vereinheitlichter Zugang zu unterschiedlichen Datenquellen (eventuell auch anderen Datenintegrationssystemen). Das Datenintegrationssystem kombiniert und vereinigt (integriert) hierzu die verschiedenen Datenquellen so, dass der Benutzer des Integrationssystems die Illusion hat, mit nur einer Datenquelle zu arbeiten. Normalerweise geschieht diese Integration, indem ein zentral administriertes globales Schema etabliert wird und anschließend semantische Abbildungen zwischen den Daten der Datenquellen und diesem globalen Schema definiert werden [LN06].

Für P2P-Datenbanken ist deshalb zunächst wesentlich, dass jeder Peer lokal ein DBMS installiert hat und Teile des Inhalts dieser lokalen Datenbank im P2P-Netzwerk zur Verfügung stellt. Die Koordination der lokalen Datenbanken der Peers sollte hierbei ohne zentrale Komponente erfolgen, um nach wie vor der Definition eines reinen P2P-Netzwerks zu genügen. Ein so charakterisiertes System wird als *P2P-Datenbank* [FKLZ04b] bezeichnet.

Bei einem *Peer Data Management System* (PDMS) [HIST03] handelt es sich um eine spezielle Untergruppe der P2P-Datenbanken. Zum Austausch der Daten müssen Abbildungen, sogenannte Mappings, zwischen den Datenschemata der beteiligten Peers erstellt werden. Die Anfrageverarbeitung erfolgt dann anhand dieser Abbildungen. Eine Anfrage wird an verbundene Peers weitergeleitet und hierbei die Anfrage gemäß der Abbildung umgeschrieben. Diese umgeschriebene Anfrage wird dann auf den verbundenen Peers ausgeführt und anschließend von diesen, mit dem selben Mechanismus, an deren Nachbarpeers weitergeleitet. Die Resultate werden beim Peer, der die Ursprungsanfrage gestartet hat, gesammelt und zusammengeführt. Da die Peers das Netzwerk jederzeit verlassen können, gibt es kein stabiles globales Schema, wie dies bei zentralen Datenintegrationslösungen der Fall ist. Stattdessen wird ein gültiges globales Schema nur während der Dauer der Anfrageverarbeitung durch die Mappings aufgebaut.

Die Einsatzgebiete für P2P-Datenbanken sind vielfältig. Aufgrund ihrer dezentralen Organisation und ihrer Flexibilität eignen sie sich besonders für ein sich schnell änderndes Umfeld. Hier sind insbesondere Forschungskollaborationen, Notfallmanagement und Anwendungsfelder, die auf einer dezentralen Organisation basieren (z.B. Semantic Web Anwendungen oder virtuelle Organisationen), zu nennen.

Der Vollständigkeit halber sei angemerkt, dass es divergierende Ansichten darüber gibt, wie Datenbanktechnologie in den Peer-to-Peer Kontext eingeführt werden kann. So gibt es

Forschungsprojekte, die Datenbanktechnologie direkt in das P2P-Netzwerk zu integrieren versuchen. Eine solche Herangehensweise wird z.B. von Bernstein u.a. [BGK+02] und auch Hellerstein u.a. [Hel03, HHB+03, HCH+05] mit dem PIER Projekt [Pie] vorgestellt. Auch das Mercury [BAS04] Projekt lässt sich in diese Kategorie einordnen. Die vorliegende Arbeit beschränkt sich ausschließlich auf die zuvor definierten P2P-Datenbanken. Die wichtigsten Forschungsarbeiten in diesem Bereich werden im Folgenden vorgestellt.

2.8.1 Allgemeine P2P-Datenbanken

Die beiden hier vorgestellten Ansätze versuchen die P2P-Architektur für eine möglichst effiziente und skalierbare Datenverteilung zu nutzen.

2.8.1.1 HiSbase

Beim HiSbase Projekt [SBG+07, SBK+07] werden die Daten eines Peers gemäß dem vorherrschenden Anfragemuster auf die anderen Peers verteilt. Ziel ist es hierbei die Verteilung der Daten so zu organisiert, dass einen Großteil der Anfrageverarbeitung lokal, d.h. ohne Weiterleitung der Anfrage an andere Peers, ausgeführt werden kann. Die Verwaltungsinformationen über die Ressourcen der Peers und der Datenverteilung werden hierbei in einer DHT gespeichert.

2.8.1.2 Armada

Armada [Gro07, Gro09] verteilt die Daten eines vorgegebenen Schemas auf die Peers des Netzwerks. Dies steht im Gegensatz zu HiSbase wo ein solches vorgegebenes Schema fehlt. Ziel der Verteilung der Daten ist es Engpässe zu vermeiden. Der einzelne Peer hat hierzu die Möglichkeit, die auf ihm gespeicherten Daten wiederum auf weitere Peers aufzuteilen. Da kein globaler Koordinator vorhanden ist erfolgt die Entscheidung über die weitere Aufteilung der Daten lokal auf dem einzelnen Peer.

2.8.2 Peer Data Management Systeme (PDMS)

PDMS können aufgrund des Vorgehens bei der Erstellung der von ihnen verwendeten Abbildungen in Systeme mit direkter und indirekter Erzeugung gegliedert werden. Letztere werden, ob ihrer Vielzahl, zusätzlich gemäß des von ihnen gewählten Verarbeitungsschwerpunktes (Abfragesprache oder Abbildungen) gegliedert.

2.8.2.1 Indirekte Erzeugung der Abbildungen

Bei der indirekten Erzeugung von Abbildungen veröffentlicht jeder Peer eine Beschreibung der eigenen Daten sowie eventuelle Abbildungen in einem gemeinsam genutzten Verzeichnis.

DBGlobe DBGlobe [PAP+03] kapselt die Daten der Peers in einen Dienst und verbreitet sie darüber. Die Daten können direkt durch XML-Abfragesprachen abgefragt werden. Voraussetzung ist jedoch eine einheitliche XML-basierte Beschreibung der Dienste und Daten. Diese Beschreibungen werden in einem strukturierten P2P-Netzwerk gespeichert [KP04].

GridVine Bei GridVine [Gri, CMAA07, CMAB+07, CMAF06, ACMHP04] werden keine Abbildungen direkt zwischen den Peers aufgebaut. Stattdessen wird ein auf P-Grid basiertes Verzeichnis (GridVine) eingeführt, das als logische semantische Overlay-Schicht fungiert. In diesem Overlay speichern die Peers Beschreibungen der von Ihnen zur Verfügung gestellten Informationen. Dies beinhaltet eine Beschreibung des eigenen Schemas und mögliche Abbildungen zu Schemata anderer Peers. Das verteilte Verzeichnis des Overlays wird auch zur Anfrageverarbeitung benutzt. Der Anfragende macht sich durch das Verzeichnis selbst ein umfassendes Bild über mögliche Datenquellen und die zum Abfragen nötigen Umformungen der ursprünglichen Anfrage. Deshalb ist kein Weiterleiten der Anfragen durch die Peers mehr nötig. Stattdessen werden diese direkt vom Anfragenden angefragt.

2.8.2.2 Direkte Erzeugung der Abbildungen

Bei dieser direkten Form der Abbildungserzeugung zwischen zwei Peer-Schemata wird unterschieden zwischen Ansätzen, bei denen die Hauptfunktionalität des Systems in den Abbildungen steckt (Piazza, coDB, Hyperion, UPDF, Self-Extending Peer Data Management und System P) und solchen, bei denen die Hauptfunktionalität in der Abfragesprache liegt (P2P XDBMS und PeerDB).

Piazza Der bekannteste Vertreter von Peer Data Management Systemen und grundlegend für die vorliegende Arbeit ist Piazza [HIM+04, TIM+03, HIST03]. Es bildet Ausgangspunkt und Basis der PDMS Forschung. Alle weiteren beschriebenen Systeme können als Varianten bzw. Erweiterungen verstanden werden.

Datenintegration wird in Piazza durch Abbildungen (Mappings) zwischen Peers erreicht. Mappings stellen hierbei formale Definitionen möglicher Antworten auf eine Anfrage über die Relation eines Peers dar. Diese Abbildungen werden in einer speziell hierfür entwickelten Sprache „Peer-Programming Language“ spezifiziert. In den Abbildungen wird genau festgelegt, wie eine Anfrage des einen Peers umformuliert werden muss, damit sie zum Schema des anderen Peers passt und somit auch dort ausgeführt werden kann.

Die Anfrageausführung läuft wie folgt ab: Eine Anfrage wird immer gegen das Schema des lokalen Peers formuliert. Entsprechend unterscheiden sich isolierte Anfragen an den lokalen Peer und Anfragen an das gesamte PDMS nicht. Anfragen an das PDMS werden an Peers weitergeleitet, die durch Abbildungen mit dem lokalen Peer verbunden sind. Beim Weiterleiten wird die Anfrage gemäß der Abbildung umgewandelt. Eine so transferierte Anfrage kann vom empfangenden Peer (wiederum über dessen Abbildungen) zu anderen Peers weitergeleitet werden. Die Ergebnisse werden entsprechend entgegengesetzter Abbildungen umformuliert und in umgekehrter Richtung zur Anfrage an den Ursprungspeer zurückgesendet.

Bei dieser Art der Anfrageverarbeitung können verschiedene Probleme auftreten. So können unterschiedliche Pfade zu einem Peer aufgrund der unterschiedlichen Abbildungen unterschiedliche Ergebnisse liefern. Des Weiteren ist die Verfolgung aller Pfade sehr ineffizient. Ansätze, um diese Probleme zu beheben, wurden von Tatarinov und Halevy [TH04] erarbeitet. Dong u.a. [DHT04] erörtert das Problem festzustellen, ob die Ergebnisse, die über einen Pfad zurückgeliefert werden, eventuell schon in den Ergebnissen eines anderen Pfades enthalten sind.

coDB Bei coDB [FKLZ04a] [FKLZ04b] werden die Schemata der einzelnen Peer Datenbanken durch Koordinationsregeln verknüpft. Für die Anfrage- und Updateverarbeitung werden

diese Koordinationsregeln (Abbildungen) in eingehende und ausgehende Verbindungen unterteilt. Eingehende Verbindungen sind Regeln, die von einem andern Peer benutzt werden, um Daten dieses Peers zu importieren. Ausgehende Verbindungen sind Regeln, die vom aktuellen Peer genutzt werden um Daten von anderen Peers zu importieren. Wenn ein Peer eine Anfrage erhält, wird sie an alle ausgehenden Verbindungen weitergeleitet. Anschließend werden Duplikate von den Ergebnissen der ausgehenden Verbindungen entfernt und die Resultate dann an die eingehenden Verbindungen weitergeleitet.

Hyperion Das Hyperion Projekt [AKK+03, KAM03a] verwendet für die Datenintegration zwischen den lokalen Datenbanken der Peers sogenannte Mapping Tables, die manuell erstellt werden müssen. Mapping Tables speichern hierbei Entsprechungen zwischen Werte-Paaren, wodurch Abbildungen zwischen den Schemata zweier Peers realisierbar werden. Zudem ist es möglich, aus bestehenden Mapping Tables Hinweise für weitere neue Verbindungen zu Mapping Tables zu gewinnen. Hierdurch kann das Erzeugen neuer und die Verwaltung bestehender Mapping Tabellen vereinfacht werden [KAM03b, KKM+03].

Unified Peer-to-Peer Database Framework (UPDF) Im von Hoschek [Hos02] entwickelten Ansatz wird das Ausführen von Anfragen in verteilten Datenbanken als Spezialfall der Ausführung einer Anfrage im Peer-to-Peer Netzwerk angesehen, bei der es keine benachbarten Peers gibt. Der Ansatz unterstützt verschiedene Techniken um Anfragepläne für PDMS zu erstellen und diese auszuführen. Die Erstellung der Abbildungen zwischen den Peers ist hingegen nicht Gegenstand des UPDF.

Self-Extending Peer Data Management Durch die Skalierung des PDMS auf die Größe des Internets entstehen erhebliche Probleme bezüglich der Datenqualität. Diesem Problem begegnet das Self-Extending Peer Data Management System [HHNR05], indem die Abbildungen zwischen den Peers durch die Verbindung mit einer Ontologie eine zusätzliche semantische Dimension besitzen. Zwischen verschiedenen Ontologien bestehen zudem wiederum Abbildungen. Bei der Anfrageausführung können durch diese Abbildungen auf Ontologien andere relevante Peers gefunden werden. Zudem können durch die Verbindungen zwischen den Ontologien wiederum Mappings zwischen den Peer Schemas erstellt werden. Die Anzahl der relevanten Mappings wird hierbei durch die zu erwartende Datenqualität bestimmt.

System P Beim System P [RN05, RN07] wird das Weiterleiten der Anfragen zu anderen Peers optimiert. Anstatt die Anfrage gemäß den Abbildungen zu allen verbundenen Peers weiterzuleiten, berechnet System P zunächst die Auswirkung dieser Weiterleitung auf das Anfrageresultat. Hierzu werden Information aus den eigenen Daten und statistische Angaben zu Abbildungen zu anderen Peers verwendet. Die Weiterleitung der Anfrage erfolgt dann nur zu Peers mit aussichtsreichen Ergebnissen.

P2P XDBMS P2P XDBMS [Zha07] verfolgt den Ansatz, XML-Datenbanken und insbesondere XQuery [XQu] so zu erweitern, dass diese von einer P2P-Architektur profitieren. Um den Umgang mit einer hochdynamischen Anzahl von unterschiedlichen Datenquellen zu erleichtern, bietet P2P XDBMS hierfür eine einheitliche Schnittstelle, die von den verschiedenen Datenquellen abstrahiert. Als Anfragesprache wird hierbei XQuery mit der XRPC [ZB07] Erweiterung für Remote Procedure Calls [BN84] benutzt.

PeerDB Bei PeerDB [NOTZ03, OST03, OTZ+03] hat jeder Peer neben der lokalen Datenbank zusätzlich ein lokales Verzeichnis in dem die Metadaten der vorhandenen Relationen gespeichert sind. Im Export Verzeichnis wird nur jener Teil des lokalen Verzeichnisses gespeichert, der anderen Peers zugänglich gemacht werden soll. Die Anfrageausführung erfolgt in zwei Phasen. Zunächst werden Agenten ausgesandt, um nach korrespondierenden Metadaten in benachbarten Peers zu suchen. Aus den so gefundenen potentiellen Datenquellen werden dann vom Benutzer die für ihn relevanten selektiert. Anschließend werden die Agenten mit der umgeschriebenen Anfrage zur Ausführung an die Peers der selektierten Datenquellen gesandt.

2.9 Sicherheit in P2P-Systemen

Sicherheit in P2P-Systemen ist aufgrund der potentiell unzuverlässigen und sich ständig ändernden Teilnehmermenge besonders kritisch. Dieser Tatsache wird mit unterschiedlichen Sicherheitsmechanismen begegnet. Für jeden der in Kapitel 2.1 erläuterten Grundpfeiler der Informationssicherheit wurden verschiedene Mechanismen entwickelt.

- Die *Verfügbarkeit*, wird wie in P2P-Systemen üblich, durch Replikation der Daten sichergestellt [APV07]. Der Gefahr, dass bösartige Knoten die Routingmechanismen von strukturierten P2P-Netzwerken manipulieren (eine gute Darstellung der möglichen Gefahren haben Sit und Morris [SM02] sowie Wallach [Wal02] zusammengestellt), begegnet man z.B. mit den Ansätzen von Castro u.a. [CDG+02] und Chen u.a. [CC07, CL06].

- *Datenintegrität* wird vornehmlich durch digitales Signieren der gespeicherten Daten gewährleistet (z.B. Past [DR01]). Darüber hinaus ist der Ansatz verbreitet, den Schlüssel des Datenobjekts dem Hashwert des Datenobjekts gleichzusetzten, der durch eine kollisionsresistente Hashfunktion berechnet wird. Dieses Verfahren, auch Self-Certifying Identifiers genannt, ermöglicht dem Empfänger, das Objekt anhand seines Objektschlüssels zu verifizieren (z.B. CFS [DKK+01]).

- Um die *Vertraulichkeit* der Daten zu wahren, kommen kryptographische Verschlüsselungsverfahren, Information Dispersal Algorithmen [Rab89] und Secret Sharing Technique [Sha79] zur Anwendung. Daneben gibt es verschiedene Ansätze zur Zugriffskontrolle in P2P-Systemen. Eine erste Hürde dieser Verfahren ist die sichere und damit eindeutige Authentifizierung der Teilnehmer (Benutzer der Peers und der Peers selbst).

2.9.1 Authentifizierung in P2P-Systemen

Die Authentifizierung von Benutzern und Peers kann auf unterschiedliche Arten erfolgen und ist stark davon abhängig, wie wichtig eine eindeutige dauerhafte Authentifizierung der Benutzer ist. Die einfachste und zugleich fragilste Form ist die Abbildung der Teilnehmer auf ihre momentane IP-Adresse. Da eine Selbstbestimmung von Benutzernamen durch die Teilnehmer Eindeutigkeitsprobleme mit sich bringt, erweist sich die Generierung der Benutzernamen über eine PKI [BEM04] (siehe Kapitel 2.3.2) als praktikabler. Dieser einfache Authentifizierungsmechanismus verhindert jedoch aufgrund der mangelnden zentralen Kontrolle nicht, dass eine physische Einheit unter verschiedenen Identitäten auftreten kann. Dies kann durch den sogenannten Sybil-Angriff [Dou01] zum Korrumpieren des gesamten Systems genutzt werden. Verhindert wird dies über die Authentifizierung mittels Zertifikate (siehe Kapitel 2.3.2.3),

wobei Zertifikate durch eine zentrale Zertifizierungsstelle vergeben werden. Da die Zertifizierungsinstanz nur einmalig zur Ausstellung der Zertifikate benötigt wird, steht diese nicht im Widerspruch zum P2P-Prinzip.

Die Ausstellung von Zertifikaten durch eine Zertifizierungsinstanz als Einstiegshürde ist für viele Anwendungsgebiete zu hoch. Oft wird sogar Wert auf die Anonymität der Benutzer gelegt. In diesen Fällen wird auf andere Mechanismen (siehe [DH06]) zurückgegriffen, um mit der unsicheren Authentifizierung der Benutzer umzugehen. So kommen Trust Management Systeme zum Einsatz, die versuchen, das Vertrauen zwischen den Benutzern zu verwalten. Die dort berechneten Vertrauenswerte können für die Authentifizierung, Autorisierung und Optimierung der Routingmechanismen verwendet werden. Meist basieren solche Systeme auf der Reputation der Peers. Die globale Reputation eines Peers wird hierbei berechnet, indem die lokale Reputation dieses Peers auf allen anderen Peers summiert wird. Durch Trust Management Systeme ist es auch möglich, bösartige Peers zu erkennen und vom Netzwerk auszuschließen. Beispiele für solche Systeme sind die Ansätze von Grothoff [Gro03], Bearly und Kumar [BK04], Gummadi und Yoon [GY04], SeAl [NT04], P2P Trust Management von der EPFL Lausanne [AD01, DA06], der EigenTrust Ansatz [KSGM03a, KSGM03b] oder PeerTrust [XL04].

2.9.2 Zugriffskontrolle in P2P-Systemen

Aufbauend auf der wie immer gearteten Authentifizierung der Benutzer und Peers existieren etliche Zugriffskontrollsysteme für P2P-Datenbanken und P2P-Systeme zur Informationsverteilung. In der vorliegenden Arbeit werden die Ansätze zur besseren Übersicht nach den von ihnen benutzten Durchsetzungsmechanismen für die Zugriffskontrolle (siehe auch Kapitel 2.5) untergliedert. Alle vorgestellten Systeme unterstützen (soweit nicht ausdrücklich anders angegeben) DAC ohne administrative Delegation. In diesem Kapitel werden Ansätze vorgestellt, bei denen Durchsetzung und Verwaltung der Zugriffskontrollregeln nicht durch eine zentrale Komponente stattfindet. Diese Grundvoraussetzung reiner P2P-Netzwerke stellt genau die Herausforderung für P2P-Zugriffskontrollsysteme dar und unterscheidet sich grundlegend von den traditionell zentralisierten Ansätzen verteilter Systeme.

2.9.2.1 Serverseitige Durchsetzung der Privilegien

P-Hera Bei P-Hera [CSMB05, MCSB05] wird unterschieden zwischen dem Besitzer einer Ressource, dem Besitzer der Daten, die im Netzwerk verfügbar sind und dem Client, der Zugriff auf diese Daten haben will. Durch die XACML Policies ist es nicht nur möglich, den Zugriff auf Ressourcen zu beschränken, sondern auch unterschiedliche Autorisierungsmechanismen für verschiedenen Aktionen auf den Objekten vorzuschreiben. Ein Datenbesitzer erstellt Zugriffsregeln die festlegen, wo Daten abgelegt werden können und welche Benutzer darauf zugreifen dürfen. Die Ressourcenbesitzer legen fest, wer Daten replizieren und bei ihnen ablegen darf.

Die Verwaltung dieser in XACML Policies formulierten Regeln übernehmen die Super Peers des P2P-Netzwerks (siehe Kapitel 2.7.1.1). Super Peers speichern dazu nicht nur die Information, welche Daten die mit ihnen verlinkten Peers besitzen, sondern auch, welche Regeln für deren Zugriff berücksichtigt werden. Hierdurch kann die Effizienz der Anfrageverarbeitung gesteigert werden. Die Super Peers werden Anfragen nur an Peers weiterleiten, deren Zugriffskontrollregeln die Anfrage auch zulassen. Die endgültige Entscheidung über die

Zulassung unterliegt jedoch immer noch dem individuellen Peer. Dies entspricht der serverseitigen Durchsetzung.

PKI-basierte Sicherheit Das System von Berket [BEM04] kommt der Idee von Benutzerkonten mit Rechten in P2P-Netzwerken sehr nahe. Voraussetzung für eine Zugriffskontrolle ist eine verlässliche Authentifizierung der Benutzer sowie eine gesicherte Kommunikation zwischen den Benutzern, um die Vertraulichkeit der Daten bei der Übertragung vom Server zum Client zu gewährleisten. Beides wird von einer Secure Group Layer Schicht [ACTT01] übernommen. Auf dieser Schicht aufbauend ist es möglich, feingranulare Zugriffsrechte basierend auf Akenti (siehe Kapitel 2.4.4.2) durchzusetzen. Die Zugriffsrechte für jedes Objekt im Netzwerk sind dabei autonom. Erhält ein anderer Peer eine Kopie eines Objekts, kann er die Zugriffsrechte für dieses Objekt selbst festlegen. Die mit Akenti ausgedrückten Zugriffsrechte werden von denjenigen Peer durchgesetzt, der das Datenobjekt gespeichert hat.

Gummadi und Yoon Der Ansatz von Gummadi und Yoon [GY04] basiert auf einem Vertrauensmodell zwischen den Knoten. Das damit berechnete Vertrauen in den anfragenden Peer wird auch zur Zugriffskontrolle herangezogen. Der Zugriff auf ein Objekt wird nur gewährt, wenn das Vertrauen in den anfragenden Peer einen gewissen Schwellenwert überschreitet. Durch Verwendung von Rollen wird die Verwaltung der Privilegien vereinfacht. Weiterführende Angaben zur Ausgestaltung der vergebenen Privilegien oder auch zum unterstützten Zugriffskontrollmodell fehlen.

Ansatz von Watanabe u.a. Beim Ansatz von Watanabe u.a. [WNH+06] werden die Zugriffsrechte von dem Peer überprüft, der die Anfrage beantwortet. Das Besondere ist dabei, dass die Peers zunächst gemäß ihrer Zugriffrechte an dem Objekt in Klassen eingeteilt werden. Hierbei entscheiden Autorisierungspeers über die Vergabe von Privilegien an andere Peers. Eine administrative Delegation von Privilegien auf andere Peers kann dadurch unterstützt werden. Die sich daraus ergebenden besonderen Vertrauensbeziehungen zwischen den Teilnehmern werden durch ein Trust Management System verwaltet.

PeGAC PeGAC [dSGBD05] ist eine RBAC-Zugriffskontrolle für Peer-to-Peer Systeme. Zugriffsregeln werden von PeGAC in XACML verfasst. PeGAC verwendet für die Zugriffskontrolle P2PSL [DGBC04], ein Sicherheitsmodul für Peer-to-Peer Netzwerke. Die Zugriffsregeln werden von den Peers selbst festlegt. Die PeGAC Zugriffskontrolle wertet diese bei einer Anfrage aus und entscheidet entsprechend auf Annahme oder Abweisung. Jeder Peer speichert hierzu die für ihn relevanten Policies lokal. Da PeGAC für eher statische Umgebungen gedacht ist, wird auf ein zentrales Register der Policies verzichtet.

PeerAccess Winslett u.a. [WZB05] haben den PeerAccess Ansatz entwickelt. Zugriffsregeln für die Peers werden als deklarative Beschreibung des Verhaltens der Peers formuliert, die Informationen holen oder geliefert bekommen. Diese Zugriffsregeln sind permanent mit den Objekten verbunden, die sie schützen sollen. Jeder Peer besitzt eine lokale Wissensbasis, in der die regelbezogenen Informationen gespeichert sind. Zudem ist darin vermerkt, welche Peers kontaktiert werden müssen, um weitergehende Informationen zu erhalten. Die Auswertung der Zugriffsregeln erfolgt jeweils lokal. Durch den Aufwand der Verwaltung der Wissensbasis ist PeerAccess eher für statische Anwendungsbereiche geeignet.

RAMARS RAMARS [JA06] ist ein Zugriffskontrollsystem für dynamische kollaborative Umgebungen. Hierbei ist die Grundannahme, dass die Benutzer der Kollaboration sich verschiedenen übergeordneten Organisationen zuordnen lassen. Während die Benutzer ständig wechseln, bleiben die übergeordneten miteinander kooperierenden Organisationen relativ statisch. Eigentümer von Datenobjekten ist entsprechend immer die Organisation und nicht ein individueller Benutzer. Aufgrund dieser Beschränkungen ist RAMARS nur sehr bedingt für P2P-Systeme einsetzbar. In RAMARS können Rollen (siehe Kapitel 2.4.3) über administrative Grenzen hinweg definiert und angewandt werden. Die Benutzer-Rollen-Zuweisung ist hierbei besonders kritisch. Um das besondere Vertrauensverhältnis zu berücksichtigen, wird das besondere Recht Delegation of Delegation Authority als Stellvertretermechanismus eingeführt. Dieses besondere Recht ermöglicht dem Eigentümer eines Objektes, Teile seiner Rollenzuweisungsberechtigungen an andere vertrauensvolle Teilnehmer weiterzugeben. Zugriffskontrollregeln werden in XACML verfasst und beim Objekteigentümer gespeichert. Hierdurch können diese nur so lange benutzt werden, wie der Objekteigentümer sich im Netzwerk befindet. Zudem entsteht dadurch ein Single Point of Failure, da die Policies einer Organisation an einem zentralen Ort verwaltet werden.

Ansatz von Zhang u. a. Zhang u.a. [ZLHL05] entwickelten eine Zugriffskontrolle für kollaborative Peergruppen. Schließen sich Peers zu einer solchen Gruppe zusammen, wird zunächst eine Collaboration Policy Instance vereinbart, die die grundlegenden Sicherheitsanforderungen der Gruppe festlegt. Die Authentifizierung der Benutzer der Gruppe kann hierbei auch nicht identitätsbasiert erfolgen. Neue Mitglieder einer Gruppe können durch die Gruppenmitglieder selbst bestimmt werden. Hierzu wird ein Wahlverfahren bei der Gründung der Gruppe festgelegt. Dasselbe Wahlverfahren kommt dann beim Gewähren von Privilegien an neue Benutzer zur Anwendung. Zugriffsregeln können in Rollen organisiert werden. Ein anfragender Peer (Client) muss mit der Anfrage auch seine Privilegien beim angefragten Peer (Server) vorlegen. Auf diesem findet dann die Evaluation der Policies statt.

2.9.2.2 Clientseitige Durchsetzung der Privilegien

Clientseitige Durchsetzungsverfahren wurden vor allem für P2P-Dateisysteme entwickelt. Die wichtigsten Projekte in diesem Bereich werden hier kurz vorgestellt.

SiRiUS SiRiUS [GSMB03] unterstützt die Aktionen Lesen und Schreiben auf Dateiebene. Jede Datei ist mit einem eigenen symmetrischen Schlüssel verschlüsselt und besitzt zudem einen weiteren Schlüssel zum Signieren der Datei. Der Schlüssel zum Signieren der Datei dient der Unterscheidung zwischen Schreib- und Leseberechtigungen. Nur ein Benutzer, der auch den Schlüssel zum Signieren der Datei besitzt, kann neue korrekte Versionen der Datei erzeugen. Um den Benutzern die Sicherheit zu geben, dass sie immer mit der neusten Version der Datei zu arbeiten, gibt es eine besondere Dateistruktur, in der die Hashwerte der aktuellen Dateien eines Benutzers aufgenommen werden. Diese muss allerdings regelmäßig durch den Benutzer aktualisiert werden. Jeder Benutzer besitzt einen privaten Schlüssel zum Verschlüsseln und einen weiteren zum Signieren. Die öffentlichen Schlüssel, die den beiden privaten Schlüsseln entsprechen sind für alle zugänglich. Ein Objekteigentümer gewährt einem Benutzer ein Privileg an einer Datei, indem er den Schlüssel der dem Privileg entspricht, mit dem öffentlichen Schlüssel des Benutzers verschlüsselt und mit der Datei speichert. Der

Benutzer kann daraufhin diesen „Dateischlüssel“ entschlüsseln und benutzen, da er im Besitz des entsprechenden privaten Schlüssels ist.

PACISSO PACISSO [Koç06, KBC07] wurde für den P2P-Objektspeicher Celeste [Cel] entwickelt. Neben Lese- und Schreibaktionen unterstützt dieser Ansatz die Definition von Objektgruppen sowie den Besitzerwechsel von Objekten. Die Zugriffskontrolle stützt sich hierbei nicht ausschließlich auf clientseitige Durchsetzung. Die Durchsetzung der Schreibberechtigungen erfolgt gruppenbasiert (siehe Kapitel 2.5.3). Hierzu wird für eine Objektgruppe eine Gruppe von Peers (hier Gatekeepers genannt) bestimmt, die für autorisierte Objektänderungen die Schlüssel- und Objektverwaltung anpasst. Erst durch diese Anpassung wird die neue Objektversion für alle anderen Peers des Netzwerks sichtbar. PACISSO verwendet eine verzögerte Zurücknahme von Privilegien. So kann ein Benutzer, dem die Privilegien zum Lesen eines Objekts entzogen wurden, ein Objekt noch lesen, solange es keine neuere Version dieses Objekts gibt. Hierdurch wird das Revocation Problem entschärft, da Objekte erst bei einer Änderung mit einem neuen Schlüssel verschlüsselt werden.

Zugriffskontrolle nach Miklau und Suciu Miklau und Suciu [MS03] sehen in einem PDMS ein Data Publishing System, weshalb die Zugriffskontrolle auf dem Client erfolgen sollte. Zugriffsrechte und -beschränkungen werden in einer Erweiterung von XQuery ausgedrückt. Durch diese wird festgelegt, welche Teile eines XML-Dokuments wie verschlüsselt werden. Vor der Veröffentlichung des Dokuments werden XQuery-Ausdrücke mit Rechteerweiterungen auf dem Dokument ausgeführt und das so verschlüsselte Dokument veröffentlicht. Das hier verwendete Zugriffskontrollmodell basiert auf dem logischen Modell zum Schutz eines XML-Baums. Die Knoten des Baums sind dabei durch Schlüssel geschützt. Somit können nur Besitzer der erforderlichen Schlüssel Zugang zu einem Element im Baum erhalten.

Ansatz von Sandhu und Zhang Der Ansatz von Sandhu und Zhang [SZ05] verwendet Trusted Computing Technology (TCT) [Tru09] zur Realisierung einer Zugriffskontrolle für P2P-Netze. Hierfür ist eine Hardwarekomponente, das Trusted Platform Modul (TPM) in den Computern aller Teilnehmer erforderlich. Diese Komponente verschlüsselt und entschlüsselt Daten, ohne den Schlüssel außerhalb der TPM zu verbreiten. Auf dem TPM basiert der clientseitige „Trusted“ Referenzmonitor, der für die Speicherung und Einhaltung der Zugriffskontrollregeln verantwortlich ist. Der „Trusted“ Referenzmonitor unterstützt hierbei auch RBAC Policies. Wie in Kapitel 2.1 dargelegt, kann durch die Verwendung von TCT der Kontrollbereich auf den Client ausgeweitet werden. Der Objekteigentümer kann entsprechend auch bei bereits auf dem Client befindlichen Daten deren Verwendung detailliert bestimmen. Da TCT die Schlüssel zur Entschlüsselung der Daten nicht preisgibt, existiert auch das Revocation Problem nicht.

Ansatz von Bouganim u.a. Bouganim u.a. [BNP04] stellen eine clientseitige Zugriffskontrolle für XML-Dokumente vor, die für P2P-Datenbanken geeignet ist. Die unerlaubte Umgehung der Zugriffskontrolle auf dem Client wird durch ein sogenanntes Secure Operating Environments verhindert. Hierbei handelt es sich um einen „Trusted“ Referenzmonitor, der auf einem TPM basiert. Die XML-Zugriffskontrollregeln bestehen aus XPath Kernelementen.

Mukkamala u.a. Beim Ansatz von Mukkamala u.a. [MCMP04] kann jeder Peer Beschränkungen für den Zugriff auf seine Ressourcen festlegen. Dies erfolgt durch Policy Statements. Die Besitzer von Objekten können Zugriffsregeln für ihre Objekte festlegen. Die Durchsetzung dieser Regeln erfolgt durch einen „Trusted“ Referenzmonitor, der auf einem TPM aufbaut und die Policies auswertet. Die für eine solche Auswertung nötigen Policies werden in einer lokalen Datenbank gespeichert. Eine Besonderheit des Ansatzes ist es, dass er auf Wunsch die Anonymität des Anfragenden schützt, indem dessen Identität durch zwischengeschaltete Knoten nicht weitergegeben wird. Dasselbe Vorgehen wird für die zurückgelieferten Objekte verwendet.

2.9.2.3 Gruppenbasierte Durchsetzung der Privilegien

Ansatz von Saxena u.a. Der Ansatz von Saxena u.a. [STY07] beschreibt eine Zugriffskontrolle für mobile Ad-hoc-Netzwerke, wobei es sich aber primär um eine Zugangskontrolle handelt. Die einzige Aufgabe der Zugriffskontrolle besteht darin, zu entscheiden, ob ein Benutzer in das Netzwerk aufgenommen wird oder nicht. Da eine weitere Abstufung der Privilegien nicht vorhanden ist, wird keines der zuvor genannten Zugriffskontrollsysteme unterstützt. Das gleiche gilt auch für einen verwandten früheren Ansatz der Autoren [STY03, NTY03].

2.10 Mängel existierender Ansätze

Es gilt nun, in Bezugnahme auf die in Kapitel 1.2 gesetzten Ziele und daraus resultierenden Anforderungen, die vorhandenen Ansätze kritisch zu durchleuchten. Sie werden überblickartig auf ihre Tauglichkeit (administrative Delegation, Unterstützung mächtiger Zugriffskontrollmodelle und Unterstützung von Datenreplikation) für den Spezialfall einer P2P-Datenbank hin betrachtet.

Allen Ansätzen ist dabei gemein, dass sie nicht speziell für den Sonderfall der P2P-Datenbanken entwickelt wurden. Ansätze, die ursprünglich für föderierte Datenbanken entwickelt wurden, lassen sich nicht, oder nur sehr beschränkt, auf P2P-Datenbanken übertragen. Im Falle der lose gekoppelten Föderationen sind die unterstützten Zugriffskontrollmodelle zu beschränkt. Da die Benutzerrechte immer den Peers und nicht den Benutzern der Peers zugewiesen werden, unterstützten sie weder RBAC noch administrative Delegation. Eng gekoppelte Föderationen sind nicht mit dem Autonomieverständnis der Peers in P2P-Systemen vereinbar und basieren zudem auf einem zentralen Koordinator, was der Definition von P2P-Systemen ebenfalls widerspricht.

Die Zugriffskontrollmechanismen für P2P-Systeme besitzen, da sie nicht speziell für P2P-Datenbanken entwickelt wurden, nicht alle wünschenswerten Eigenschaften. So bietet keines dieser Systeme die Nutzung schon vorhandener lokaler Zugriffskontrollregeln an, da in allgemeinen P2P-Systemen vielmals keine lokale Zugriffskontrollkomponente auf den Peers vorhanden ist. Im Allgemeinen sind Zugriffskontrollsysteme mit serverseitiger Durchsetzung gut für P2P-Datenbanken geeignet. Durch die Unterstützung ausdrucksstarker Zugriffskontrollmodelle und der Möglichkeit, Teile der administrativen Tätigkeiten der Privilegienverwaltung an andere Benutzer zu delegieren (administrative Delegation), sind sie auch für große Systeme mit vielen Benutzern geeignet. Da die Durchsetzung der Zugriffskontrolle durch den individuellen Besitzer der Daten erfolgt, ist diese über das Netzwerk verteilt und unterliegt keiner zentralen Kontrolle. Problematisch ist dies allerdings in Bezug auf die Datenreplikation. Wenn Datenobjekte in diesem Umfeld repliziert werden, muss auch die Durchsetzung

der Privilegien dieser replizierten Datenobjekte durch den Peer vorgenommen werden, der diese speichert (Replikatserver). Dies ist in einem P2P-System nicht praktikabel, in dem das Vertrauen zwischen den beteiligten Peers beschränkt ist.

Beim Betrachten der einzelnen Ansätze, fallen über diese Grundproblematik hinaus weitere Probleme auf.

Der P-Hera Ansatz ist durch die individuell geregelte Authentifizierung zwar sehr flexibel, allerdings fehlt dadurch ein einheitlicher Standard über die Zuverlässigkeit der Authentifizierung. Dies ist insbesondere bei administrativer Delegation problematisch, da dadurch Privilegien an nicht authentifizierte Benutzer weitergereicht werden können. Zudem ist die Granularität der administrativen Delegation auf ganze Policies beschränkt. Auf diese Weise ist ein Benutzer entweder berechtigt, die gesamte Zugriffsregel oder gar nichts zu ändern. Abstufungen wie jene, dass ein Benutzer nur neue Objekte oder Benutzer zur Zugriffsregel hinzufügen darf, existieren nicht.

Der Ansatz PKI-basierte Sicherheit nimmt die Problematik mit der Replikation geschickt auf und gibt dem Replikationsserver zusammen mit dem Datenobjekt sämtliche Rechte am replizierten Datenobjekt. Mehr noch, jeder Benutzer, der das Datenobjekt bekommt (z.B. durch eine Anfrage), erhält damit zeitgleich alle Rechte an diesem Objekt. Der Ansatz beugt damit falsch vorgespielter Sicherheit im Fall von Datenreplikation vor, ersetzt die Sicherheit aber durch die Abgabe aller Kontrolle über das Objekt, was nicht zur Behebung des Replikationsproblems beiträgt. Die Speicherung der Akenti Privilegien ist ein kritischer Punkt. Aus den Ausführungen von Berket geht nicht hervor, ob diese zentral registriert werden oder vollständig repliziert an allen benötigten Peers vorliegen.

Der Ansatz von Gummadi und Yoon unterstützt keinerlei administrative Delegation von Privilegien. Zudem erscheint es zumindest problematisch, etwas Relatives wie Vertrauen in absoluten Zahlen zu messen und zu bewerten. Zudem ist Vertrauen nie kontextfrei. Einen allgemeinen Vertrauenswert anzunehmen, erscheint deshalb nicht sinnvoll. Noch kritischer ist es, diesen Vertrauenswert als Kriterium für die Vergabe von Privilegien zu wählen.

Der Ansatz von Watanabe u.a. ermöglicht die administrative Delegation von Privilegien. Durch die grobe Einteilung der Benutzer in Klassen ist allerdings keine feingranulare administrative Delegation möglich. Das zwischen den Peers benötigte Vertrauen muss deshalb dementsprechend groß sein. Dies erscheint selbst bei der Verwendung eines Trust Management Systems eher unrealistisch.

Der PeGAC Ansatz ist sehr flexibel und unterstützt ausdrucksstarke Zugriffskontrollmodelle. Eine administrative Delegation von Privilegien ist allerdings nicht vorgesehen. Dies ist zudem lediglich in der Granularität einer ganzen Policy denkbar. Aufgrund der Organisation dieser Policies erscheint der Ansatz wie bereits erwähnt eher für statische Umgebungen geeignet. Ähnliches gilt für den PeerAccess Ansatz. Darüber hinaus fehlt dort die Möglichkeit der administrativen Delegation von Privilegien.

Bei RAMARS ist zwar eine administrative Delegation von Privilegien möglich, allerdings verlangt RAMARS hierbei für die Rechteverwaltung sehr statische Teilnehmermenge. Dies verhindert die Nutzung von RAMARS in allgemeinen P2P-Systemen.

Im Ansatz von Zhang u.a. ist immer eine Gruppe zusammen für die Zulassung von neuen Peers und die Weitergabe von neuen Privilegien zuständig. Diese Idee erscheint auf den ersten Blick vielversprechend. Eine administrative Delegation von Privilegien auf einzelne Peers ist allerdings nicht vorgesehen. Durch die Unterstützung mächtiger Zugriffskontrollmodelle kann dieser Ansatz auch für größere Netzwerke eingesetzt werden.

Ansätze, die auf einer clientseitigen Durchsetzung der Privilegien basieren, lösen das Pro-

blem der Datenreplikation. Indem die Daten verschlüsselt auf dem Replikatserver gespeichert werden, muss kein besonders großes Vertrauen zu diesem bestehen. Allerdings hat die clientseitige Durchsetzung andere systembedingte Nachteile, wie z.B. das Revocation Problem (siehe Kapitel 2.5.2). Die systembedingt beschränkte Unterstützung mächtiger Zugriffskontrollmodelle ist offensichtlich.

Der SiRiUS Ansatz unterstützt zwar Schreib- und Leseaktionen nicht jedoch erweiterte DAC-Modelle noch RBAC. Zudem ist administrative Delegation von Privilegien nicht möglich. Die Verwaltung der Privilegien ist sehr aufwändig, da die Schlüssel für jeden Benutzer einzeln verschlüsselt gespeichert werden müssen. Dieser Ansatz scheint deshalb für P2P-Datenbanken nicht geeignet.

Bei PACISSO ist die Schlüsselverwaltung und damit die Verwaltung der Privilegien effizienter gelöst. Durch die Möglichkeit, Objekte in Gruppen aufzuteilen, kann auch der administrative Aufwand der Privilegienzuteilung reduziert werden. Die administrative Delegation von Privilegien ist jedoch unmöglich; und auch die Unterstützung mächtigerer Zugriffsmodelle fehlt. Die Granularität der Zugriffsrechte ist, wie für ein Dateisystem sinnvoll, auf Dateien festgelegt, was für Datenbanken allerdings zu grob ist. Das nur verzögert mögliche Entziehen der Privilegien birgt ebenfalls Probleme. In P2P-Systemen, die primär der Informationsweitergabe dienen, und in denen somit eher selten Datenänderungen vorkommen, könne Privilegien damit faktisch nie entzogen werden.

Der Ansatz von Miklau und Suciu kann ebenfalls weder administrative Delegation von Privilegien noch RBAC unterstützen. Jedoch ist hier die Granularität der Zugriffsrechte nicht länger auf Dateien beschränkt, sondern kann sehr flexibel an die jeweiligen Bedürfnisse angepasst werden. Dem steht die fehlende Unterstützung von Schreibaktionen entgegen.

Die Ansätze von Sandhu und Zhang, von Mukkamala u.a. sowie jener von Bouganim u.a. bieten zwar ebenfalls keine Möglichkeit der Delegation von Privilegien, beheben aber das Revocation Problem, das eines der größten Probleme von clientseitiger Durchsetzung darstellt, und unterstützen mächtige Zugriffskontrollmodelle. Allerdings ist hierfür Voraussetzung, dass jeder Peer ein Trusted Platform Modul (TPM) besitzt und darauf aufbauend einen „Trusted“ Referenzmonitor. Dies ist jedoch kritisch, da so in die Autonomie des einzelnen Peers eingegriffen wird, was als zwingende Voraussetzung zu restriktiv ist.

Beim Ansatz von Saxena u.a. handelt es sich um den einzigen mit einer gruppenbasierten Durchsetzung. Allerdings kann kaum von Zugriffskontrolle gesprochen werden, sondern vielmehr von Zugangskontrolle. Ein solcher „Alles oder Nichts“ Ansatz ist für P2P-Datenbanken ungeeignet.

Die Ergebnisse finden sind in Tabelle 2.1 nochmals zusammengefasst. Die Ansätze wurden hier bezüglich der Anforderungen verglichen wobei RBAC stellvertretend für alle mächtigeren Zugriffskontrollmodelle steht. Ein + in der Spalte bedeutet der Ansatz erfüllt diese Anforderung und − entsprechend die Nichterfüllung einer Anforderung. Bei ∘ konnte das Ergebnis aufgrund fehlender Information nicht eindeutig festgestellt werden und (+) in der P2P-Spalte wird vergeben falls sich der Ansatz wie bei PeGAC und PeerAccess auf eine statische Umgebung verlässt oder wie bei Sandhu, Bouganim und Mukkamala besondere Bedingungen an die Peers (hier eine TPM) gestellt werden. Die Spalte Lokal gibt an, ob eine Nutzung der von lokalen Zugriffskontrollregeln der Peers möglich ist und die Spalte „Böse Peers“ bewertet ob die Sicherheit des Systems auch bei bösartigen Peers im Netzwerk gegeben ist.

Zusammenfassend lässt sich sagen, dass Zugriffskontrollsysteme, die mächtige Zugriffskontrollmodelle wie RBAC unterstützen, eine serverseitigen Durchsetzung der Privilegien verfolgen. Zudem unterstützen diese Systeme die administrative Delegation von Privilegien

Ansatz	Admin. Delegation	RBAC	Granularität	Replikation	P2P	Lokal	Böse Peers
Serverseitige Durchsetzung							
P-Hera	+	+	−	−	+	−	−
PKI	−	−	+	+	∘	−	−
Gummadi	−	+	∘	−	+	∘	+
Watanabe	+	−	+	−	∘	−	+
PeGAC	−	+	+	−	(+)	−	∘
PeerAccess	−	−	∘	−	(+)	−	∘
RAMARS	+	+	+	−	−	−	∘
Zhang	−	+	+	−	+	−	+
Clientseitige Durchsetzung							
SiRiUS	−	−	−	+	+	−	+
PACISSO	−	−	−	+	+	−	+
Miklau	−	−	+	+	−	∘	∘
Sandhu	−	+	+	+	(+)	−	+
Bouganim	−	+	+	+	(+)	−	+
Mukkamala	−	+	−	+	(+)	−	+
Gruppenbasierte Durchsetzung							
Saxena	−	−	−	+	+	−	+

Tabelle 2.1: Vergleich bestehender Zugriffskontrollsysteme

durchweg nur sehr grobgranular, was für P2P-Datenbanken ungenügend ist. Systeme, die die Replikation in P2P Umgebungen unterstützen, basieren auf der clientseitigen Durchsetzung. Zudem gibt es keinen Ansatz, der die Nutzung schon bestehender Zugriffsregeln der lokalen Systeme berücksichtigt. Aus der Tatsache, dass keiner der bisher bekannten Systeme die feingranulare administrative Delegation von Privilegien, sowie die Möglichkeit der Koordination (und damit Benutzung) bestehender Zugriffskontrollsysteme bietet, folgt die Entwicklung des Peer Access Control System (PACS) in der vorliegenden Arbeit. Dies wird in den folgenden Kapiteln ausführlich beschrieben.

Kapitel 3

Dezentral koordinierte Zugriffskontrolle (PACS)

In diesem Kapitel wird das „Peer Access Control System“ (PACS) vorgestellt. Es unterscheidet sich grundlegend in Herangehensweise und Aufbau von den zuvor vorgestellten Zugriffskontrollsystemen. Kapitel 3.1 gibt einen Überblick über den grundsätzlichen Aufbau von PACS. Besonderes Augenmerk liegt hierbei auf den Bestandteilen und deren Zusammenspiel. Die vorgestellten Komponenten werden dann in den Kapiteln 3.2 bis 3.8 ausführlich beschrieben. Zunächst erfolgt jedoch die Erläuterung wichtiger Termini:

Benutzer Ein Benutzer gehört in PACS zu genau einem Peer. Ein Peer kann mehrere Benutzer haben.

Datenobjekte Ein Datenobjekt ist eine Abstraktion der von PACS unterstützten Datenformen. D.h., es kann unter anderem ein Tupel einer Datenbanktabelle oder auch eine Datei repräsentieren. Datenobjekte gehören ebenfalls zu genau einem Peer und können diesem eindeutig zugeordnet werden.

Dateneigentümer Dateneigentümer eines Datenobjekts ist der Peer, der dessen Speicherung im Netzwerk veranlasst hat. Der Peer ist hierbei Stellvertreter für seine Benutzer.

Datenbesitzer Datenbesitzer eines Datenobjekts ist jeder Peer, der dieses Datenobjekt lokal gespeichert hat. Dies ist insbesondere bei der Datenreplikation der Fall. Der Datenbesitzer hat keine Rechte am Datenobjekt.

3.1 PACS – eine Übersicht

PACS basiert, im Gegensatz zu den in Kapitel 2.9.2 vorgestellten P2P-Zugriffskontrollsystemen, auf der Koordination bestehender lokaler Zugriffskontrollkomponenten der Teilnehmer. Diese Grundidee wurde vom Autor bereits in [Stu06] dargestellt. Zielsetzung hierbei ist die umfassende Koordination der lokalen Zugriffskontrollkomponenten, ohne die Autonomie der Peers einzuschränken. Zudem sollte diese Koordination möglichst wenig Aufwand erzeugen.

Aufgrund der Heterogenität der lokalen Peers ist hierzu zunächst die Etablierung eines gemeinsamen Austauschformats für die Informationen der Zugriffskontrollkomponenten erforderlich. Hierzu veröffentlichen die Peers den für das Netzwerk relevanten Teil ihrer lokalen

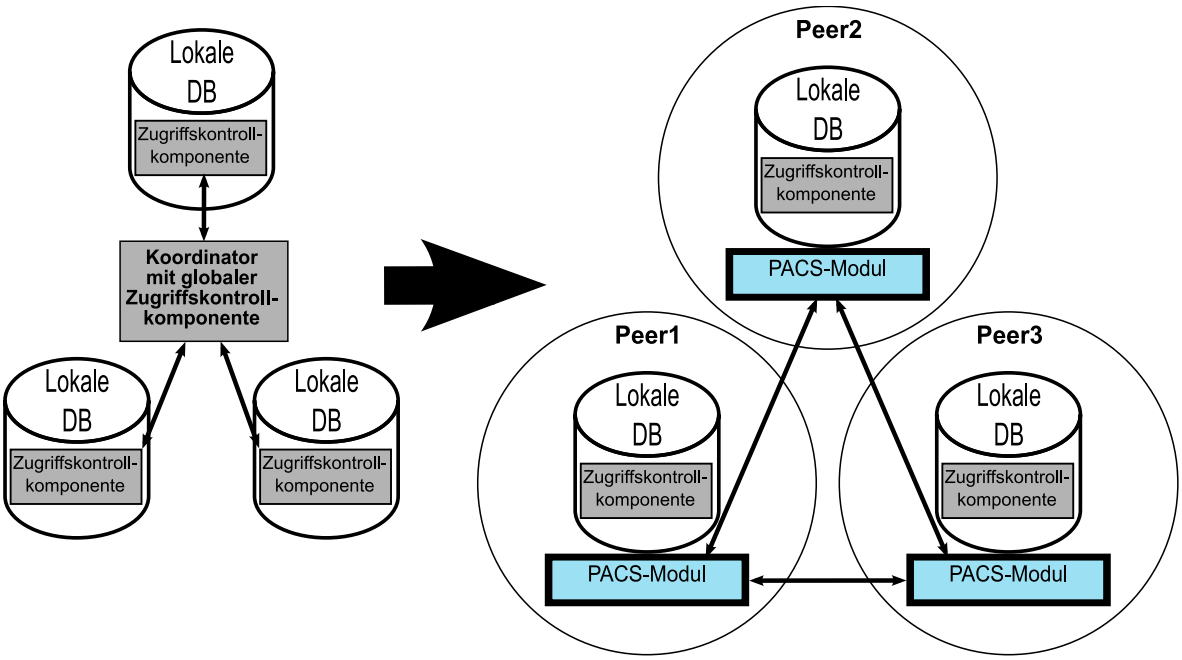


Abbildung 3.1: Überführung der zentralen in die dezentrale Koordination

Zugriffskontrollregeln durch sogenannte Export Policies. Als Austauschformat wird hierbei XACML (siehe Kapitel 2.4.4.1) verwendet, das um Angaben zur administrativen Delegation erweitert wurde. Die Export Policy eines Peers enthält die Benutzer des Peers, die Datenobjekte die vom Peer im Netzwerk angeboten werden und Privilegien der Benutzer des Peers auf den Datenobjekten. Da die Erstellung dieser Export Policies von den Peers lokal durchgeführt werden, wird dieser Teil der Zugriffskontrolle auch als lokale Ebene bezeichnet.

Die in der lokalen Ebenen enthaltenen Zugriffskontrollinformationen, werden nun zur Etablierung der globalen Zugriffskontrolle der globalen Ebene benutzt. Die Koordination der lokalen Zugriffskontrollkomponenten in PACS erfolgt durch das Gewähren von Privilegien zwischen Benutzern und Datenobjekten verschiedener Peers. Wird einem Benutzer von Peer p ein Privileg an einem Objekt des Peers p' gewährt und $p \neq p'$ dann wird dies als globales Privileg bezeichnet. Globale Privilegien verbinden die Zugriffskontrollkomponenten zweier Peers, da bei ihnen Objekt und Benutzer verschiedenen Knoten angehören. Durch die Vergabe solcher globaler Privilegien wird eine globale Zugriffskontrolle etabliert. Der Aufbau von PACS erfolgt deshalb von unten nach oben (Abbildung 3.1), beginnend bei den lokalen Zugriffskontrollkomponenten der Teilnehmer, die dann durch Koordination derselben die globale Zugriffskontrolle etablieren. Das globale Zugriffskontrollmodell unterstützt hierbei sowohl RBAC als auch administrative Delegation. Durch diese Koordination besteht die Möglichkeit, bestehende Zugriffskontrollkomponenten und die darin enthaltenen Zugriffskontrollregeln weiter zu verwenden, womit einer der Mängel schon bestehender Systeme eliminiert wird.

Im Gegensatz zu föderierten Datenbanken (siehe Kapitel 2.6) wird hierbei der zentrale Koordinator durch die direkte dezentrale Koordination zwischen den Peers ersetzt. Dies ist in Schaubild 3.1 dargestellt. Während im zentralen Fall (links dargestellt) alle Zugriffskontrollkomponenten mit einem globalen Koordinator verbunden sind erfolgt die Koordination bei PACS (rechts dargestellt) durch die direkte Koordination zwischen den lokalen PACS-

Modulen.

Die Aufgaben des zentralen Koordinators werden umfassend durch PACS übernommen:

- Verwalten der globalen Privilegien
- Speichern der globalen Privilegien
- Durchsetzung der gespeicherten globalen Privilegien

Das globale Zugriffskontrollmodell von PACS unterstützt hierbei sowohl RBAC als auch administrative Delegation. Zur Verwaltung und insbesondere Speicherung der globalen Privilegien benötigt PACS hierzu einen gemeinsam genutzten, verteilten globalen Privilegienspeicher. Dieser muss verlässlich sein, um die Datensicherheit der gespeicherten Privilegien auch dann zu garantieren falls sich bösartige Peers im Netzwerk befinden. In PACS werden hierzu zwei Alternativen entwickelt. Der DHT-Privilegienspeicher basiert auf einer DHT. Die DHT selbst und das darunterliegende Chord Protokoll werden hierbei erweitert, um die Datensicherheit trotz bösartiger Peers im Netzwerk zu ermöglichen. Beim unstrukturierten P2P-Privilegienspeicher wird hingegen ein unstrukturiertes P2P-Netzwerk mittels aktiver Replikation in einer definierten Replikationsgruppe zu einem verlässlichen Privilegienspeicher erweitert.

Neben der Verwaltung der Privilegien ist PACS auch für deren Durchsetzung verantwortlich. Auch hierbei wurden von PACS zwei alternative Möglichkeiten, nämlich die serverseitige und die clientseitige Durchsetzung, entwickelt. Die gewählte Durchsetzungsvariante entscheidet über die Organisation der Privilegien in den zwei alternativen Privilegienspeichern. Während bei der serverseitigen Durchsetzung die Herausforderung in der geschickten Organisation der Privilegien im Privilegienspeicher liegt, kommt bei der clientseitigen Durchsetzung noch die Organisation der Schlüsselverwaltung hinzu.

Um RBAC und administrative Delegation bei der clientseitigen Durchsetzung zu unterstützten wird in PACS, die Schlüsselverwaltung durch eine Stellvertretergruppe und schwellenkryptographischer Verfahren durchgeführt. Jeder Dateneigentümer etabliert hierbei eine Gruppe von Stellvertretern die, falls er sich nicht im Netzwerk befindet, die korrekte Durchsetzung der Privilegien an seinen Datenobjekten garantieren. Für die autonome Verwaltung der Stellvertretergruppe kommen ebenfalls schwellenkryptographische Verfahren zum Einsatz.

3.1.1 Integration von PACS in P2P-Anwendungen

Für die Integration von PACS in P2P-Anwendungen sind zwei grundlegende Architekturen denkbar: die Realisierung als Middleware (siehe Abbildung 3.2) und die Realisierung als Anwendungsmodul (siehe Abbildung 3.3). Der Vorteil der Middleware-Lösung ist ihr integrierter Ansatz. Die Sicherheitsmechanismen kapseln den Datenzugriff, was deren Umgehung erschwert und Sicherheitslecks verhindert. Zudem erfolgt die Tätigkeit von PACS für die P2P-Anwendungen transparent. Ein Nachteil der Middleware-Lösung ist die Festlegung auf eine konkrete P2P-Netzwerkinfrastruktur (z.B. Pastry oder Chord). Dies beschränkt die durch PACS unterstützten Anwendungen auf jene, die auf derselben Netzinfrastruktur basieren wie PACS. Aufgrund der Vielzahl von Netzinfrastrukturen ist eine solche Festlegung nicht sinnvoll. Hinzu kommt, dass lediglich die Durchsetzung der Privilegien transparent ist, wohingegen die Privilegienverwaltung außerhalb der Applikation realisiert werden muss. Anders ist dies in der Modul-Lösung. Hier ist die Privilegienverwaltung inklusive der Speicherung der Privilegien im PACS-Modul integriert. Dabei ist deren konkrete Realisierung für den Benutzer transparent. Dies ermöglicht auch die Trennung des Datenspeichers für Anwendungsdaten

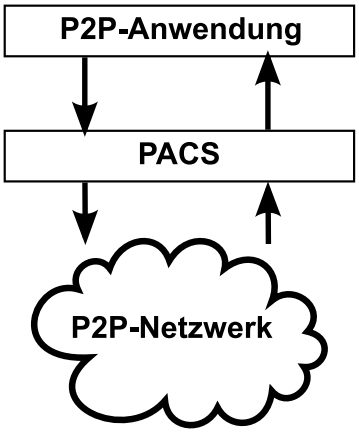


Abbildung 3.2: PACS als Middleware

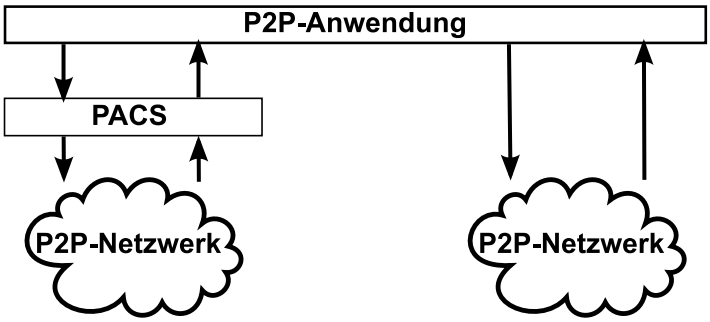


Abbildung 3.3: PACS als Anwendungsmodul

und Privilegien. Eine solche Trennung ist wünschenswert, da das Speichern von Privilegien im Allgemeinen höhere Sicherheitsanforderungen an den Datenspeicher stellt als das Speichern von Anwendungsdaten. Nachteil der Modul-Lösung ist, dass sie bei einem Datenzugriff keine Transparenz bietet, weshalb eine Anpassung der P2P-Anwendungen nötig ist. Zudem benötigt PACS Zugriff auf die gespeicherten Anwendungsdaten, da Änderungen der Privilegien Datenänderungen verursachen können (z.B. erneute Verschlüsselung der Daten mit einem neuen Schlüssel beim Entziehen von Privilegien bei clientseitiger Durchsetzung). Die Anwendung muss PACS hierzu die von der verteilten Hashtabelle (DHT) bekannten Operationen *put(key,value)* zum Speicheren der Daten und *get(key)* zum Abfragen der gespeicherten Daten zur Verfügung stellen.

Trotz der genannten Probleme der Modul Lösung überwiegen deren Vorteile, insbesondere in Bezug auf den wesentlich breiteren Einsatzbereich. Aus diesem Grund wurde PACS als Modul realisiert.

3.1.2 Der Aufbau des PACS-Moduls

Die Architektur von PACS wurde entsprechend seinen Hauptaufgaben *Authentifizierung*, *Privilegienverwaltung* mit *Privilegienspeicher* und *Durchsetzung der Zugriffskontrolle* gegliedert (siehe Abbildung 3.4).

Die *Authentifizierung* von Benutzern und Peers ist die Grundvoraussetzung für die durch PACS etablierte Zugriffskontrolle. Die *Privilegienverwaltung* ist für alle Aufgaben des Peers zuständig, die die Vergabe bzw. den Entzug von globalen Privilegien betreffen. Da die globalen Privilegien letztlich auf den Export Policies basieren, sind diese Bestandteil der Privilegienverwaltung. Der Inhalt der Export Policy stellt gleichzeitig die Verbindung zur lokalen Zugriffskontrollkomponente des Peers dar. Auch die Auswertung der Privilegien ist in die Privilegienverwaltung integriert. Da die administrative Delegation von Privilegien unterstützt wird, ist die Überprüfung des Delegationspfades für die Privilegien Bestandteil dieser Überprüfung. Die Speicherung der von PACS verwalteten globalen Privilegien übernimmt der *Privilegienspeicher*.

In Abbildung 3.4 sind die zwei konkreten Realisierungen des Privilegienspeichers dargestellt. Die Speicherung der Privilegien erfolgt dort zum einen mit Hilfe einer DHT und zum

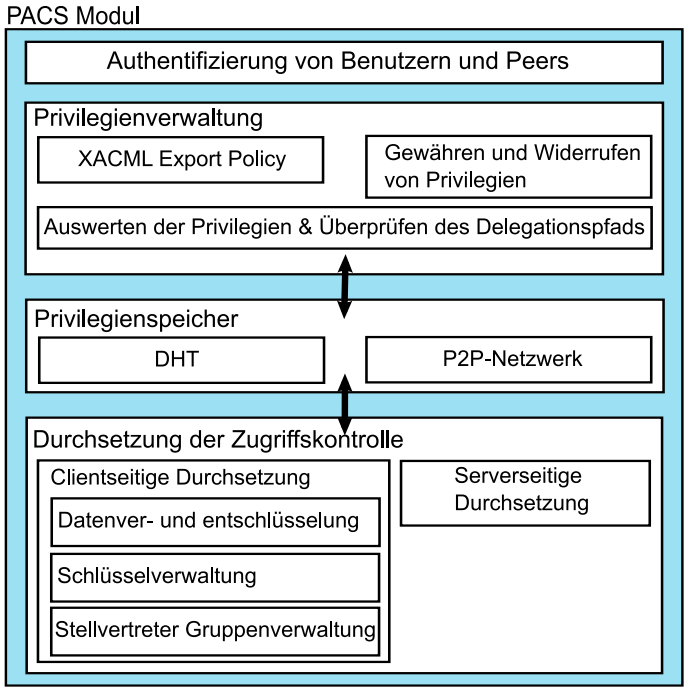


Abbildung 3.4: Aufbau des PACS-Moduls

anderen mittels eines unstrukturierten P2P-Netzwerks (siehe Kapitel 3.5.7 bzw. 3.5.6). Bei beiden in PACS integrierten Privilegienspeicher werden die Daten unter den Teilnehmern verteilt. Hierzu bedarf es der Kommunikation der PACS-Module untereinander. Die zu speichernden Privilegien werden vor dem Speichern von der Privilegienverwaltung überprüft.

Die *Durchsetzung der Zugriffskontrolle* benötigt Zugriff auf den Privilegienspeicher. Besitzt der anfragende Benutzer die für eine Anfrage benötigten Privilegien, ist er zur Durchführung der Anfrage autorisiert. Bei der konkreten Durchsetzung unterstützt PACS die serverseitige sowie die clientseitige Variante. Beide unterscheiden sich in der von ihnen unterstützten Funktionalität (siehe Kapitel 3.7 und 3.8).

3.1.3 Aufbau eines P2P-Netzes mit PACS

In einem P2P-Netzwerk mit PACS besitzt jeder Peer ein PACS-Modul (siehe Abbildung 3.5). Die vom PACS-Modul verwaltete Export Policy stellt die Verbindung zum lokalen Zugriffskontrollsystem des jeweiligen Peers dar und ist zudem die Basis der von PACS verwalteten globalen Privilegien. Durch den Austausch der Export Policies zwischen den Peers etabliert sich ein verteiltes Benutzer- und Privilegienverzeichnis. Hierzu kommuniziert das PACS-Modul direkt mit PACS Modulen anderer Peers. Des Weiteren braucht das PACS-Modul zur Durchsetzung der Zugriffskontrolle Zugriff auf die Anwendungsdaten der P2P-Applikation. Das P2P-Modul stellt hierfür die beiden Operationen *put(key,value)* und *get(key)* zur Verfügung. Bei dem P2P-Modul handelt es sich um eine Abstraktion der durch die P2P-Anwendung implementierten Funktionalität, welche Zugriff auf die lokale Datenbasis hat.

Der Zugriff des PACS-Moduls auf die lokale Datenbasis und insbesondere die lokale Zugriffskontrolle wird von PACS für die Durchsetzung und Verwaltung der globalen Zugriffskontrolle benötigt. Nur so ist es möglich, die Export Policy von der lokalen Zugriffskontrolle

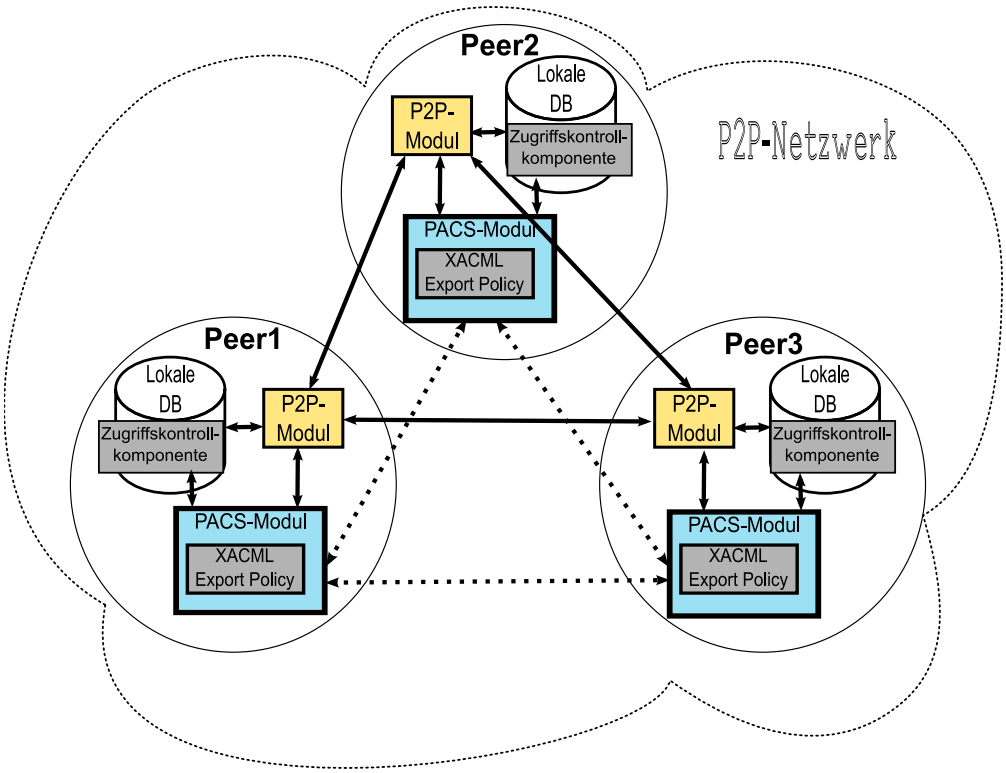


Abbildung 3.5: Aufbau eines P2P-Netzwerks mit PACS

abzuleiten und Anfragen an lokale Datenobjekte zu prüfen.

Die PACS-Module kommunizieren für die verteilte Speicherung und Durchsetzung der Privilegien direkt miteinander. Sie bilden somit ein unabhängiges P2P-Netzwerk im schon bestehenden P2P-Netz.

3.1.4 Beispielanwendung

Zum besseren Verständnis wird die Funktionsweise von PACS im Folgenden mittels einer Beispielanwendung konkretisiert. Diese wird hier in ihren Grundzügen dargestellt. In den folgenden Kapiteln wird sie immer dann herangezogen, wenn eine Konkretisierung auf einen Einzelfall das Verständnis vertieft.

Ausgangspunkt der Beispielanwendung sind kollaborierende Forschungsgruppen, die gemeinsam an einem Forschungsprojekt arbeiten. Hierzu ist es notwendig, den Teilnehmern untereinander Zugriff auf die lokalen Forschungsdaten der einzelnen Gruppen zu gewähren. Da die Forschungskooperation nur zeitlich befristet ist und ständigen Änderungen obliegt, wird auf den Aufbau einer zentralen Infrastruktur verzichtet. Stattdessen etabliert die Forschungskooperation ein Peer Data Management System (PDMS) zwischen den Teilnehmern. Aufgrund der zum Teil sicherheitsrelevanten Daten des Forschungsprojekts wird eine Zugriffskontrolle für das PDMS benötigt. Da die Sicherheitsanforderungen sich ständig verändern, wird auf eine effiziente Verwaltung der Benutzerprivilegien besonders Wert gelegt. Insbesondere die administrative Delegation der Privilegienverwaltung an ausgesuchte Teilnehmer des Netzwerks soll eine schnelle und effiziente Adaption der Benutzerprivilegien an neue Gege-

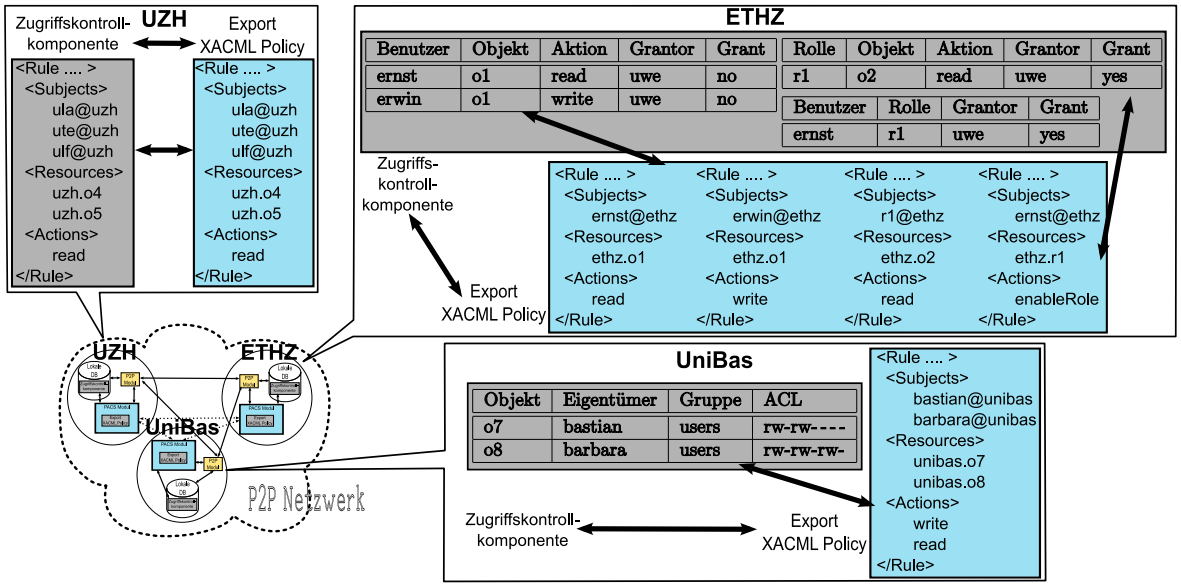


Abbildung 3.6: Beispielanwendung

benheiten ermöglichen.

Um die Beschreibung der Anwendung zu erleichtern, ohne jedoch die Allgemeingültigkeit zu verlieren, erfolgt eine Beschränkung auf drei Forschungsgruppen der Forschungsgemeinschaft: Die Universität Zürich (UZH), die Universität Basel (UniBas) und die ETH Zürich (ETHZ). Alle Forschungsgruppen benutzen unterschiedliche lokale Datenbanken und setzen entsprechend verschiedene Zugriffskontrollmechanismen ein. Diese Beispielkonstellation ist in Abbildung 3.6 dargestellt. Die UZH benutzt lokal ein XACML-basiertes Zugriffskontrollsystem für ihre lokale Datenbank. Die Überführung dieser Privilegien in die XACML-basierte Export Policy ist entsprechend einfach. Die ETHZ verwendet hingegen ein traditionelles Datenbanksystem, das die Privilegien in Form mehrerer Tabellen speichert. Neben einer Privilegientabelle existiert dort zusätzlich eine Rollentabelle mit den Zuordnungen der Privilegien zu den Rollen sowie eine Tabelle mit Benutzerzuordnungen für diese Rollen. Die Grantor-Spalte enthält den Benutzer, der ein Privileg vergeben hat; die Grant-Spalte zeigt an, ob ein Privileg vom Benutzer ebenfalls weitergegeben werden darf. Auch diese Privilegien wurden für die Export Policy vom Peer ETHZ in XACML Policies übersetzt. Selbiges gilt für UniBas. Allerdings setzt UniBas die von Dateisystemen bekannte „Access Control List“ (ACL) ein. Jedes Datenobjekt besitzt dabei seine eigene ACL. Darin sind alle Benutzer mit entsprechenden Privilegien für dieses Objekt vermerkt. Die ACL in diesem Beispiel hat das aus Unix bekannte Format mit ACL Kategorien Besitzer, Gruppe und Alle. Alle Benutzer der Gruppe *users* und der Objekteigentümer *bastian* selbst haben beispielsweise Lese- und Schreibzugriff auf Objekt $o7$ (siehe Abbildung 3.6).

Die Benutzerin $ula@uzh$ leitet ein Teilprojekt und benötigt hierzu lesenden Zugriff auf das Objekt $unibas.o7$. Das hierfür erforderliche Privileg erhält sie vom Benutzer $bastian@unibas$. Da es sich bei $bastian@unibas$ um einen lokalen Benutzer von $UniBas$ handelt, bezieht er das Recht zur Vergabe dieses lesenden Zugriffs aus den in der Export Policy von UniBas enthaltenen Privilegien. Dabei gibt $bastian@unibas$ auch das Privileg zur Weitergabe dieses Privilegs an $ula@uzh$ weiter (administrative Delegation). $ula@uzh$ arbeitet in dem Teilprojekt

Benutzer	Objekt	Aktion	Grantor	Grant	Zeit
ula@uzh	unibas.o7	read	bastian@unibas	yes	1
ernst@ethz	unibas.o7	read	ula@uzh	no	2
r1@ethz	uzh.o4	read	ula@uzh	yes	3

Tabelle 3.1: Globale Privilegien des Beispielszenarios

Benutzer	Rolle	Grantor	Grant	Zeit
ula@uzh	ethz.r1	ernst@ethz	yes	4
bastian@unibas	ethz.r1	ula@uzh	no	5

Tabelle 3.2: Globale Benutzerzuordnungen des Beispielszenarios

mit *ernst@ethz* zusammen. Da dieser für eine erfolgreiche Zusammenarbeit ebenfalls den Zugriff auf Objekt *unibas.o7* benötigt, gewährt *ula@uzh ernst@ethz* das Leserecht an diesem Objekt.

Die ETHZ hat für ein anderes Teilprojekt die Rolle *ethz.r1* eingeführt. Da *ula@uzh* an diesem Teilprojekt mitarbeitet, weist ihr *ernst@ethz* die Rolle *ethz.r1* zu. Um *ula@uzh* zu ermöglichen diese Rolle auch weiteren Projektteilnehmern zuzuweisen, vergibt Ernst die Rolle mit administrativer Delegation (Grantoption) weiter. *ula@uzh* nutzt diese Möglichkeit um *bastian@unibas* ebenfalls diese Rolle zuzuweisen, allerdings ohne Grantoption. Da dieses Teilprojekts auch noch Zugriff auf Daten der UZH benötigt, gibt *ula@uzh* der Rolle *ethz.r1* das Privileg für lesenden Zugriff auf Objekt *uzh.o4*. Somit können alle Mitglieder der Rolle *ethz.r1* auch auf *uzh.o4* lesend zugreifen. Die soeben beschriebenen Vorgänge lassen sich in der Privilegientabelle (Tabelle 3.1) und Benutzerzuordnungstabelle (Tabelle 3.2) ablesen. Aus diesem Beispielszenario ergeben sich folgende Fragen:

- Wie erfolgt die Authentifizierung der Benutzer und Peers?
- Welche Eigenschaften muss das Zugriffskontrollmodell besitzen um die gewünschte Funktionalität zu gewährleisten?
- Wo und wie werden die globalen Privilegien gespeichert?
- Wie werden die Privilegien verwaltet?
- Wie erfolgt die Durchsetzung der gespeicherten Privilegien?

3.2 Authentifizierung in PACS

Grundvoraussetzung für die Zugriffskontrolle ist die sichere Authentifizierung von Benutzern und Peers [FKC07]. Die Authentifizierung basiert in PACS auf einer PKI mit zentraler Zertifizierungsinstanz (siehe Kapitel 2.3.2). Deshalb ist es erforderlich, dass jeder Peer und jeder Benutzer eines Peers ein gültiges Zertifikat besagter Zertifizierungsinstanz besitzt. Die Verwendung einer zentralen Zertifizierungsinstanz für die teilnehmenden Peers hat zwei Gründe. Zum einen wird dadurch die Eindeutigkeit der Benutzer- und Peernamen garantiert. Zum anderen stellt die Zertifizierungsinstanz sicher, dass jeder Peer nur eine einzige Identität annehmen kann. Hierzu werden den Peer eindeutig kennzeichnende Informationen in das Zertifikat

aufgenommen. Nur durch die sichere Verhinderung multipler Identitäten von Peers, kann das Netzwerk gegenüber Sybil-Angriffen [Dou01] geschützt werden.

Neben der Authentifizierung wird die PKI in PACS zur sicheren Kommunikation zwischen Benutzern und Peers herangezogen. Dies garantiert die Vertraulichkeit der Daten während des Transports zwischen den Benutzern bzw. Peers. Sofern nicht anders vermerkt, werden bei den folgenden Beschreibungen die zu schützenden Daten wie z.B. Schlüssel, Datenobjekte und Abstimmungsergebnisse immer in verschlüsselter Form zwischen Peers und Benutzern ausgetauscht.

3.3 Das Zugriffskontrollmodell von PACS

Die Zugriffskontrolle erfolgt in PACS auf zwei Ebenen: *lokal* und *global*. Die *lokale Ebene* betrifft hierbei den individuellen Peer und wird von diesem autonom verwaltet. Die *globale Ebene* beinhaltet alle Aufgaben und Maßnahmen, die peerübergreifend durchgeführt werden. Entsprechend dieser zwei Ebenen gibt es in PACS *lokale Privilegien* und *globale Privilegien*. Lokale Privilegien betreffen nur einen Peer und werden nur auf der lokalen Ebene verwaltet. Bei globalen Privilegien sind hingegen mehrere Peers involviert, weshalb deren Verwaltung auf der globalen Ebene erfolgt. Im Folgenden werden die Aufgaben der beiden Ebenen, die Verbindung zwischen ihnen und die jeweils verwendeten Zugriffskontrollmodelle vorgestellt.

3.3.1 Lokale Ebene und Export Policies

Durch die lokale Ebene ist gewährleistet, dass die bestehenden Zugriffsregeln in die globale Zugriffskontrolle überführt werden können. Da PACS hierzu die lokalen Zugriffskontrollkomponenten der Peers verwendet, ist zunächst die Heterogenität dieser Komponenten zu überwinden. Wie in Kapitel 2.4 dargelegt gibt es eine Vielzahl verschiedener Zugriffskontrollmodelle. Um diese zu koordinieren ist deshalb eine Überführung dieser Zugriffskontrollmodelle in ein gemeinsames Zugriffskontrollmodell nötig. Dies ist ein bekanntes Problem föderierter Datenbanken [JD94, dVFS07]. Aufgrund des fehlenden globalen Koordinators im P2P-Umfeld erfolgt die Übersetzung der lokalen Zugriffskontrollregeln in das gemeinsame Zugriffskontrollmodell durch den individuellen Peer. Die Erzeugung dieser *Export Policy* ist somit Bestandteil des PACS-Moduls.

Die Export Policy erfüllt in PACS drei Aufgaben:

1. Sie definiert ein gemeinsames Austauschformat für die Privilegien.

2. Sie legt die Privilegien und Rollen der lokalen Benutzer für die veröffentlichten Daten des Peers fest.

3. Sie spezifiziert grundlegenden Einschränkungen für die veröffentlichten Daten.

Aufgrund der Tragweite der hier vorgenommenen Entscheidungen und der hierzu benötigten Rechte erfolgt die Erstellung dieser Export Policies durch einen Benutzer mit entsprechend umfassenden Zugriffsrechten für die lokalen Datenquelle (z.B. dem Datenbankadministrator oder Systemadministrator).

3.3.1.1 Gemeinsames Austauschformat

Wie bereits ausgeführt, ist zunächst die Überführung der lokalen Zugriffskontrollregeln in ein gemeinsames Zugriffskontrollmodell nötig. Hierbei ist zuerst ein geeignetes gemeinsames Zugriffskontrollmodell für PACS festzulegen. In PACS wird XACML als gemeinsames Zugriffskontrollmodell benutzt [SDZ08]. Die Gründe hierfür sind vielfältig. So ist XACML aufgrund der Möglichkeit verschiedene XACML Policies miteinander zu kombinieren sehr flexibel und ausdrucksstark. Durch diese Ausdrucksstärke werden DAC-Modelle wie auch RBAC von XACML unterstützt. Dies erleichtert die Übersetzung der lokalen, in das gemeinsame Zugriffskontrollmodell. Zudem ist seine offene und standardisierte Struktur sehr gut als Austauschformat geeignet. Im Folgenden wird zugunsten der Lesbarkeit jeweils nur von einer Export Policy pro Peer gesprochen, wenngleich jede Export Policy aus mehreren XACML Policies bestehen kann. Ist von mehreren Export Policies die Rede, handelt es sich um Export Policies von verschiedenen Peers (wobei wiederum jeder Peer nur eine Export Policy besitzt).

Die Überführung der lokalen Zugriffskontrollregeln in XACML Policies ist am einfachsten, wenn die lokale Zugriffskontrollkomponente schon auf XACML basiert. Für die am weitesten verbreiteten Datenquellen und ihre Zugriffskontrollkomponenten können für die Erstellung der Export Policies von PACS Konverter angeboten werden. Wichtig ist bei der Erstellung und Pflege der Export Policies, dass ihr Inhalt konsistent zum Inhalt der lokalen Zugriffskontrollkomponente ist. Insbesondere bei Änderungen lokaler Privilegien muss überprüft werden, ob nicht auch die XACML Policy entsprechend angepasst werden muss. Wenn z.B. einem Benutzer das Privileg zum Lesen eines Objektes entzogen wird, ist darauf zu achten, dass dem Benutzer dieses Privileg auch in der XACML Export Policy entzogen wird. Die Übersetzung der Zugriffsregeln in XACML Policies ist unkompliziert; Benutzer und Rollen werden auf XACML subjects, Objekte auf XACML resources und Aktionen auf XACML actions abgebildet. Die Export Policy legt weiterhin die Rollen und deren Privilegien fest. Für jede Rolle wird hierbei nach XACML-Standard eine Role Policy angelegt. Diese Policy dient der Prüfung der Benutzerzuordnung. Die Privilegien einer Rolle werden in der Permission Policy festgelegt. Die Benutzerzuordnung selbst findet in der XACML Role Assignment Policy statt. Ein vereinfachtes Beispiel einer Export Policy für den Benutzer *ernst@ethz*, der Leseberechtigung für das Objekt *ethz.o*1 besitzt, ist in Abbildung 3.7 dargestellt.

Wie bereits in Kapitel 2.4.4.1 ausgeführt, gibt es im XACML-Standard keine Möglichkeit festzulegen, wer Policies ändern, löschen oder erstellen darf. Der Vorschlag von Rissanen und Firozabadi [RF04] zur Einführung von administrativen XACML Policies behebt diesen Mangel. Diese administrativen Policies legen fest, wer eine Policy verändern, erstellen und löschen darf. Sie sind Bestandteil des derzeitigen Working Draft für XACML Version 3.0 [R$^+$07]. Da in diesem nur die Erstellung ganzer Policies behandelt wird, besteht keine Möglichkeit, nur teilweise Veränderungen von bestehenden Policies zu reglementiert. Dies liegt daran, dass sich die Regeln administrativer Policies immer auf die gesamte XACML Policy beziehen und nicht auf ihre jeweiligen Bestandteile. Die Integration der Informationen zur administrativen Delegation in die XACML Policies ist deshalb nicht offensichtlich. Da diese Angaben jedoch zentraler Bestandteil von PACS sind, wurden für PACS drei Vorschläge entwickelt, diese in XACML integrieren zu können.

Aufteilen der Privilegien Eine einfache Lösung der Integration von administrativer Delegation in XACML, besteht darin die XACML Policies in Privilegien zu unterteilen. Je-

```
<Rule RuleId="export:ethz" Effect="Permit">
<!--
   <Grantor>
      <Subject>
         <SubjectMatch MatchId="&function;rfc822Name-equal">
            <AttributeValue
               DataType="&datatype;rfc822Name">admin@ethz
            </AttributeValue>
            <SubjectAttributeDesignator
               AttributeId="&subject;subject-id"
               DataType="&datatype;rfc822Name"/>
         </SubjectMatch>
      </Subject>
   </Grantor>
   <RuleGrantOption DataType="&xml;string">yes</RuleGrantOption>
   <Timestamp></Timestamp>
-->
   <Subject>
      <SubjectMatch MatchId="&function;rfc822Name-equal">
         <AttributeValue
            DataType="&datatype;rfc822Name">ernst@ethz
         </AttributeValue>
         <SubjectAttributeDesignator
            AttributeId="&subject;subject-id"
            DataType="&datatype;rfc822Name"/>
      </SubjectMatch>
   </Subject>
   <Resource>
      <ResourceMatch MatchId="&function;anyURI-equal">
         <AttributeValue
            DataType="&xml;anyURI">ethz.o1</AttributeValue>
         <ResourceAttributeDesignator
            AttributeId="&resource;resource-id"
            DataType="&xml;anyURI"/>
      </ResourceMatch>
   </Resource>
   <Action>
      <ActionMatch MatchId="&function;string-equal">
         <AttributeValue
            DataType="&xml;string">read</AttributeValue>
         <ActionAttributeDesignator
            AttributeId="&action;action-id"
            DataType="&xml;string"/>
      </ActionMatch>
   </Action>
</Rule>
```

Abbildung 3.7: Auszug aus einer Export Policy mit erweiterten Angaben zur administrativen Delegation

des Privileg entspricht anschließend einer eigenen XACML Policy. So können administrative XACML Policies für die einzelnen XACML Policies erstellt und die administrative Delegation in XACML integriert werden. Der Nachteil dieser Lösung ist, dass die Anzahl der XACML Policies explodiert, da jede dieser XACML Policies eine eigene administrative XACML Policy benötigt. Hinzu kommen eventuell noch die Policies für Benutzerzuordnung und Zuordnung der Privilegien bei RBAC. Dies macht die Erstellung und Administration der Export Policies sehr umständlich, weshalb diese Lösungsmöglichkeit in PACS verworfen wurde.

Erweiterung der administrativen Policies Ein alternativer Ansatz erweitert die administrativen Policies. Die Regeln der administrativen XACML Policies müssen sich nun anstatt auf eine ganze Policy auf die einzelnen Regeln der Policy beziehen. Da eine Regel einem Privileg entspricht ist die Aufteilung der XACML Policies nicht mehr nötig. Für eine solche Erweiterung müssen die in der administrativen Policy enthaltenen Regeln die einzelnen Regeln der XACML Policy referenzieren können. Erst dann kann die administrative Policy die Einschränkungen, die bei einer Erstellung einer neuen Regel gelten sollen, spezifizieren.

Daneben müssen administrative Regeln bei einer Auswertung der Policies überprüft werden können. Eine solche Auswertung macht allerdings eine Änderung des bei XACML festgelegten Policy Evaluationsprozesses nötig. Der XACML Working Draft [R$^+$07] zu administrativen Policies legt fest, dass für jede Policy, der Ersteller autorisiert werden muss. Hierzu wird eine administrative XACML-Anfrage erzeugt, die dann gegenüber den vorhandenen Policies evaluiert wird. Nur Policies, die entsprechend autorisiert sind, nehmen anschließend am Evaluationsprozess für die ursprüngliche Anfrage teil. Bei der Realisierung der Erweiterung muss diese Überprüfung dann nicht für jede Policy, sondern für jede ihrer Regeln durchgeführt werden. Hierzu ist eine inhaltliche Prüfung der Policy erforderlich, die beim Policy Evaluationsprozess nicht vorgesehen ist. Wünschenswert ist hier zudem nur die Autorisierung der tatsächlich in einer Entscheidung involvierten Regeln zu überprüfen um den Aufwand dieser Überprüfung zu minimieren. Solch tiefgreifende Änderungen an der XACML-Spezifikation und am XACML-Evaluationsprozess erscheinen für die geringe zusätzliche Funktionalität und Ausdrucksstärke von XACML jedoch wenig sinnvoll. Deshalb wurde auch diese Lösungsmöglichkeit für PACS verworfen.

Überprüfung der administrativen Delegation außerhalb von XACML Auf der Basis der bisherigen Überlegungen nimmt der dritte Lösungsansatz die Informationen zur administrativen Delegation zwar in die XACML Policies auf, lässt diese aber nicht von XACML auswerten. In der XACML Policy wird festgelegt, wem es erlaubt ist, das durch die entsprechende Regel spezifizierte Privileg an andere Benutzer weiterzugeben.

Einerseits ist diese Lösung in ihrer Flexibilität gegenüber den zuvor vorgestellten Varianten eingeschränkt. Andererseits ist die Überprüfung der korrekten Weitergabe der Privilegien wesentlich einfacher. Nur wenn diese Überprüfung bestanden wurde, erfolgt die XACML Policy Evaluation. Hierzu wird der anfragende Benutzer durch den lokalen Benutzer an der Wurzel des Delegationspfades ersetzt, um eine erfolgreiche Evaluation zu ermöglichen. In unserem Beispielszenario führt bei einer Anfrage von *ula@uzh* an die UniBas für das Objekt *unibas.o*7 der Delegationspfad zu *bastian@unibas*. Die XACML-Evaluation wird entsprechend mit dem Benutzer *bastian@unibas* durchgeführt. Da so die Zusammenarbeit zwischen Benutzern unterstützt und für die Benutzer wenig Aufwand erzeugt wird, ist diese Lösung ein guter Kompromiss zwischen Flexibilität und Benutzerfreundlichkeit. Um diese Lösung zu

realisieren werden drei zusätzliche Attribute für das XACML Rule Element benötigt. Diese sind:

Grantor (<*Grantor*>): Grantor ist der Benutzer, der das Privileg vergeben hat.

Grantoption (<*RuleGrantOption*>): Die Grantoption legt fest, ob das Privileg an andere Benutzer weitergegeben werden darf.

Zeitstempel (<*Timestamp*>): Der Zeitstemple enthält den Zeitpunkt, zu dem das Privileg gewährt wurde.

Die für die Einführung dieser Attribute auf Regelebene nötige Erweiterung der XACML-Spezifikation wird vermieden, indem diese Informationen als XML-Kommentare dargestellt werden, die bei der XACML Policy Evaluation ignoriert werden. So ist keine Anpassung der XACML Policy Evaluation Implementierung nötig. Die Aufgabe der Zusatzattribute besteht darin, die Privilegien der administrativen Delegation zu dokumentieren und so die Überprüfung des Delegationspfades zu ermöglichen. Eine um diese administrativen Attribute ergänzte XACML Policy ist in Abbildung 3.7 zu sehen.

Die administrativen Attribute haben auch für die Regelelemente der XACML Role Assignment Policy Bedeutung. In diesem Fall gewähren sie dem betreffenden Benutzer administrative Privilegien für die Benutzerzuordnung zu dieser Rolle.

Verbindung der Export Policy zur lokalen Zugriffskontrolle Die Export Policy wird vom PACS-Modul verwaltet. Seine Erzeugung muss allerdings von einem Benutzer mit den entsprechenden Rechten auf der lokalen Datenquelle (z.B. Systemadministrator oder Datenbankadministrator) veranlasst werden. Die in der Export Policy enthaltenen Informationen sind nur relevant für Anfragen vom P2P-Netzwerk. Deshalb ist die Export Policy immer nur ein Auszug aus den lokalen Zugriffskontrollregeln der lokalen Zugriffskontrollkomponenten. Entsprechend werden lokale administrative Entscheidungen auf den lokalen Zugriffskontrollregeln getroffen. Falls durch diese Entscheidungen auch Privilegien der Export Policy betroffen sind, ist es nötig, diese Änderungen zu propagieren.

Neu von einem Benutzer eines Peers hinzugefügte Datenobjekte müssen von diesem zunächst in die Export Policy aufgenommen werden. Erst dann ist die Privilegienverwaltung für ein Objekt mit PACS möglich. Das Hinzufügen eines neuen Objekts in die Export Policy wird von PACS überwacht. Solche Veränderungen der Export Policy können, wie die Erstellung einer Export Policy, nur von einem Benutzer mit entsprechend umfassenden lokalen Rechten erfolgen. Dies gilt auch für das Hinzufügen von Benutzern oder Rollen.

3.3.2 Globale Ebene

Die globale Zugriffskontrolle wird durch die Koordination der Privilegien der Export Policies etabliert. Diese Koordination geschieht in PACS durch die Vergabe *globaler Privilegien*. Globale Privilegien sind die Bausteine der globalen Zugriffskontrolle, deren Verwaltung und Durchsetzung von PACS auf der globalen Ebene übernommen wird.

Alle in den Export Policies enthaltenen administrativen Privilegien bilden die Grundmenge an Privilegien der globalen Zugriffskontrolle. Die Export Policy eines Peers kann administrative Privilegien besitzen, die sich auf die als XACML-Regeln definierten Privilegien der Export Policy beziehen. PACS geht dabei von der Annahme aus, dass jede XACML-Regel eindeutig

durch die Kombination lokaler Benutzer, Aktion und Objekt bestimmt werden kann. Im weiteren Verlauf wird der Benutzer in der XACML Policy als „lokaler Benutzer“ bezeichnet, da Export Policies nur Privilegien für lokale Benutzer des Peers enthalten. Diese lokalen Benutzer mit entsprechenden administrativen Privilegien stellen die Verbindung zwischen der lokalen und globalen Ebene her, indem sie *globale Privilegien* an Benutzer anderer Peers vergeben. Vor der Erteilung solcher Privilegien steht die Festlegung des globalen Zugriffskontrollmodells.

3.3.2.1 Das globale Zugriffskontrollmodell

Aufgrund der in Kapitel 1.2 formulierten Anforderungen ist die Unterstützung erweiterten Zugriffsmodelle und administrativer Delegation auf globaler Ebene erforderlich. In PACS wurde deshalb RBAC gewählt, das mit DAC kombiniert wurde.

Die von PACS gewählte Kombination ist wie folgt. Jeder Benutzer ist ein Benutzer nach dem RBAC-Standard, wodurch er Mitglied verschiedener Rollen ist. Durch diese Mitgliedschaft erhält er die Privilegien, die den Rollen zugewiesen sind. Mit DAC ist es daneben möglich, einem Benutzer in PACS weitere individuelle Privilegien zu gewähren. Auf diese Weise lässt sich eine spezifische Privilegienvergabe realisieren, ohne neue Rollen einführen zu müssen. Insbesondere kann PACS dadurch auch ohne die Verwendung von Rollen betrieben werden, falls z.B. die lokale Zugriffskontrollkomponenten eines Peers RBAC nicht unterstützt oder nicht verwendet. Die Rollenadministration wird in PACS durch administrative Delegation verteilt. So können auch Rollen administrative Privilegien erhalten.

Das Zugriffskontrollmodell besitzt lediglich positive Privilegien (siehe Kapitel 2.4.1). Entsprechend sind alle Zugriffe, die nicht durch eine Regel autorisiert sind, nicht erlaubt. Der Ausschluss negativer Privilegien erleichtert die Auswertung der Zugriffskontrollregeln und vermeidet Konflikte zwischen positiven und negativen Privilegien.

Eine weitere Besonderheit von PACS ist die Hinzunahme des lokalen Benutzers in ein Privileg. Diese Information erleichtert die Auswertung der Privilegien, da die Abbildung eines Privilegs auf die zugrundeliegende XACML Policy Regel ohne Auswertung des Delegationspfades möglich ist.

In Abbildung 3.8 sind in einem erweiterten Gegenstand-Beziehungs-Diagramm die Zusammenhänge zwischen den passiven Einheiten der Zugriffskontrolle in PACS vollständig dargestellt. Wie ersichtlich, können sowohl Benutzer als auch Rollen Privilegien zugewiesen bekommen. Ein Privileg ist die Beziehung zwischen Aktion, Objekt, Zeitstempel, Rechteempfänger (Grantee) und dem Benutzer, der das Recht vergibt (Grantor). Die Aktion ist hierbei noch über eine „lokaler Benutzer“ Beziehung mit dem Benutzer verbunden. Ein Benutzer kann Mitglied einer Rolle sein. Die Mitgliedschaft in einer Rolle wird von einem Grantor gewährt und besitzt wie das Privileg einen Zeitstempel. Rollen können Hierarchien bilden, wobei eine Rolle immer nur von einer Elternrolle abgeleitet werden kann.

Die Beziehung zwischen Subjekten und Benutzern ist in Diagramm 3.9 dargestellt. Ein Subjekt gehört immer zu genau einem Benutzer, jedoch kann ein Benutzer mehrere Subjekte besitzen. Ein Subjekt kann aktivierte Rollen zugewiesen bekommen. Eine Rolle kann zu mehreren Subjekten gehören.

Die formale Beschreibung dient der klaren Spezifikation des für PACS entwickelten Zugriffskontrollmodells inklusive seiner Regeln. Die Notation ist dabei an Ferraiolo u.a. [FKC07] angelehnt, erweitert um die DAC-Darstellung von Biskup [Bis08] und Jonscher [Jon98]. Ein Überblick über wichtige Funktionen des Zugriffskontrollmodells gibt Tabelle 3.3.

Im PACS-Zugriffskontrollmodell gibt es die Grundmengen $BENUTZER$, $SUBJEKTE$,

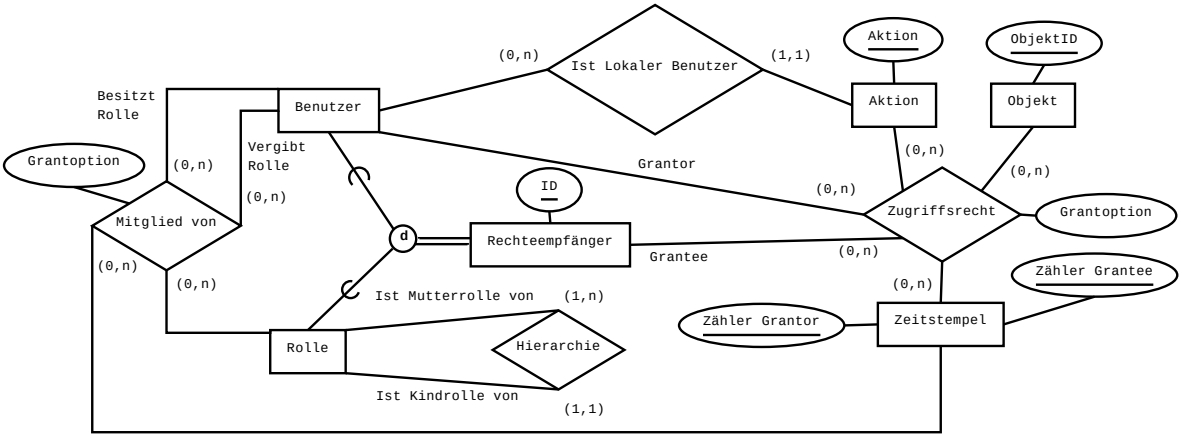


Abbildung 3.8: Zugriffskontrollmodell als erweitertes Gegenstands-Beziehungs-Diagramm

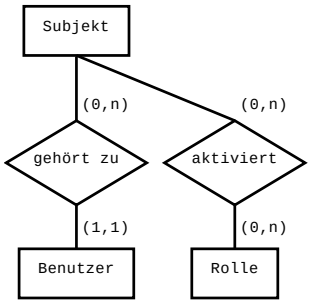


Abbildung 3.9: Beziehungen zwischen Benutzer und Subjekt

rollen_benutzer	Ermittelt die Menge der zu einer Rolle zugewiesenen Benutzer
benutzer_rollen	Ermittelt die Menge der Rollen, die einem Benutzer zugewiesen wurden
rollen_priv	Ermittelt die einer Rolle zugewiesene Menge an Privilegien
subjekt_benutzer	Zuordnung des Subjekts auf den ihm zugehörigen Benutzer
subjekt_rollen	Ermittelt die für ein Subjekt aktivierten Rollen
benutzer_priv	Ermittelt die Menge der Privilegien die einem Benutzer direkt zugewiesen wurden (DAC)
autorisiert	Überprüft ob eine Anfrage eines Subjekts autorisiert ist
lokaler_benutzer_objekt	Ermittelt, ob der übergebene Benutzer denselben Peer als Ursprung hat wie das übergebene Objekt
lokaler_benutzer_rolle	Ermittelt, ob der übergebene Benutzer denselben Peer als Ursprung hat wie die übergebene Rolle
grant_priv	Überprüft, ob der Benutzer über ein globales administratives Privileg verfügt, das ihn zur Weitergabe des angegebenen Privilegs berechtigt
grant_rolle	Überprüft, ob der Benutzer über ein globales administratives Privileg verfügt, das ihn zur Weitergabe der angegebenen Benutzerzuordnung berechtigt
zugr_benutzer	Ermittelt alle Zugriffsrechte eines Benutzers
check_zugr	Überprüft den Delegationsgraphen des angegebenen Zugriffsrechts
check_bz	Überprüft den Delegationsgraphen der angegebenen Benutzerzuordnung

Tabelle 3.3: Wichtige Funktionen des Zugriffskontrollmodells

$ROLLEN$, $AKTIONEN$ und $OBJEKTE$, die entsprechend die Menge aller Benutzer, Subjekte, Rollen, Aktionen und Objekte bezeichnen. Für die Verwaltung der administrativen Delegation kommt die Menge $GO = \{w, f\}$ (mit w = wahr und f = falsch) der Grantoption und die Menge aller Zeitstempel TS hinzu.

Die Menge der Benutzerzuordnungen BZ ist eine $n : m$ Beziehung zwischen Benutzern und Rollen. Diese Beziehung wird für die administrative Delegation um die Angaben von Grantor, Grantoption und Zeitstempel erweitert. Der Grantor entspricht hierbei wiederum der Menge der Benutzer.

$$BZ \subseteq BENUTZER \times ROLLEN \times BENUTZER \times GO \times TS$$

Zur Vereinfachung werden Projektionsfunktionen für die Benutzerzuordnung eingeführt. $b(bz)$ gibt den Benutzer, $r(bz)$ die Rolle, $go(bz)$ die Grantoption, $ts(bz)$ den Zeitstempel und $g(bz)$ den Grantor der Benutzerzuordnung $bz \in BZ$ zurück. Die Menge der zu einer Rolle r zugewiesenen Benutzer ermittelt die Funktion $rollen\_benutzer$. Die Menge der Rollen, die einem Benutzer b zugewiesen wurden, liefert die Funktion $benutzer\_rollen$.

$$\begin{aligned} &rollen\_benutzer(r \in ROLLEN) \rightarrow \{BENUTZER\} \\ &rollen\_benutzer(r) = \{b \mid b \in BENUTZER \wedge (b, r, g, go, ts) \in BZ\} \end{aligned}$$

$$\begin{aligned} &benutzer\_rollen(b \in BENUTZER) \rightarrow \{ROLLEN\} \\ &benutzer\_rollen(b) = \{r \mid r \in ROLLEN \wedge (b, r, g, go, ts) \in BZ\} \end{aligned}$$

Die Menge der Privilegien $PRIV$ ist die Beziehung zwischen der Aktion, dem lokalen Benutzer, dem Objekt, dem Grantor, der Grantoption und dem Zeitstempel. Hier ist zu beachten, dass der *lokale Benutzer* lediglich der genaueren Spezifizierung der Aktion dient und *nicht* dem Benutzer gleichzusetzen ist, dem ein Privileg zugewiesen wird.

$$PRIV = AKTIONEN \times BENUTZER \times OBJEKTE \times BENUTZER \times GO \times TS$$

Auch für Privilegien werden Projektionsfunktionen eingeführt. $a(p)$ gibt die Aktion, $lb(p)$ den lokalen Benutzer, $o(p)$ das Objekt, $go(p)$ die Grantoption, $ts(p)$ den Zeitstempel und $g(p)$ den Grantor eines Privilegs zurück. Die Zuordnung der Privilegien PZ ist definiert als:

$$PZ \subseteq ROLLEN \times PRIV$$

Die einer Rolle zugewiesene Menge an Privilegien ermittelt die Funktion:

$$\begin{aligned} &rollen\_priv(r \in ROLLEN) \rightarrow \{PRIV\} \\ &rollen\_priv(r) = \{p \mid p \in PRIV \wedge (r, p) \in PZ\} \end{aligned}$$

Die Menge der Subjekt Benutzer Zuordnungen SB ist wie folgt definiert.

$$\begin{aligned} &SB \subseteq SUBJEKTE \times BENUTZER \\ &\forall (s, b) \in SB : \forall (s, b') \in SB : b = b' \end{aligned}$$

Ein Subjekt s kann auf den ihm zugehörigen Benutzer eindeutig abgebildet werden. Dies geschieht durch die Funktion:

$$\begin{aligned} &subjekt\_benutzer(s \in SUBJEKTE) \rightarrow BENUTZER \\ &subjekt\_benutzer(s) = \{b \mid b \in BENUTZER \wedge (s, b) \in SB\} \end{aligned}$$

Die aktivierten Rollen des Subjekts s können durch die Funktion *subjekt_rollen* bestimmt werden.

$$subjekt\_rollen(s \in SUBJEKTE) \rightarrow ROLLEN$$
$$subjekt\_rollen(s) = \{r \mid r \in ROLLEN \wedge (subjekt\_benutzer(s), r) \in BZ\}$$

Hierbei ist zu beachten, dass für einen Benutzer b mit Subjekt s, $subjekt\_rollen(s) \subseteq benutzer\_rollen(b)$ gilt, da nicht alle Rollen von b in s aktiviert sein müssen.

PACS unterstützt die im RBAC-Standard festgelegte hierarchische Vererbung der Rollen. Deshalb muss das Modell um diese erweitert werden. Die Vererbungsbeziehung wird hierbei als $\rightarrow$ dargestellt. Die Vererbung der Rollen selbst kann als azyklischer Graph dargestellt werden, mit den Rollen als Knoten und $\rightarrow$ als gerichtete Kanten. $a \rightarrow b$ bezeichnet hierbei die Generalisierung, d.h. b ist eine Generalisierung von a (oder b ist die Superklasse von a, bzw. a ist die Spezialisierung von b). Die Beziehung $\rightarrow$ bezeichnet hierbei die Vererbungsbeziehung von Benutzerzuordnungen *und* Zuordnungen der Privilegien. Entsprechend gilt $a \rightarrow b$ nur, wenn alle Privilegien der Rolle b auch Bestandteil von Rolle a sind und alle Benutzer von Rolle a auch der Rolle b angehören. Eine Vererbungsbeziehung zwischen zwei Rollen muss nicht immer direkt sein; $\rightarrow^*$ bezeichnet hierzu die transitive Vererbungsbeziehung. $a \rightarrow^* b$ gilt dann, wenn $a \rightarrow q_1 \ldots q_n \rightarrow b$ für $n \geq 1$. Dieselbe Terminologie wird auch für die Benutzerzuordnung und die Zuordnung von Berechtigungen verwendet. Benutzer b ist entsprechend einer Rolle r zugeordnet, wenn $b \rightarrow r$. Hingegen ist b für r autorisiert, falls $b \rightarrow^+ r$ gilt, wobei $\rightarrow^+$ die transitive Benutzerzuordnung von $\rightarrow$ darstellt.

Die Rollenhierarchie $RH \subseteq ROLLEN \times ROLLEN$ ist eine partielle Ordnung der Rollen, eben die Vererbungsbeziehung. Von dieser Hierarchie können Autorisierungspfade abgeleitet werden, die die lineare Ordnung der Rollen mit zunehmender Generalisierung enthalten. Die einzelnen Rollen sind hierbei mit der direkten Vererbungsbeziehung $\rightarrow$ verbunden. Ein Autorisierungspfad, dargestellt als $\succeq$, ist deshalb eine Sequenz von Rollen, die durch direkte Vererbungsbeziehungen miteinander verbunden sind. Der Pfad $r_1 \succeq r_n$ bezeichnet den Autorisierungspfad von $r_1 \rightarrow^* r_n$ und beinhaltet entsprechend die Sequenz der Rollen $\langle r_1, \ldots, r_n \rangle$. Die Menge aller Autorisierungspfade AP ist wie folgt definiert.

$$\langle r_1, \ldots, r_n \rangle \in AP \Leftrightarrow \{\forall i \in \{1, \ldots, (n-1)\} \mid r_i \rightarrow r_{i+1}\}$$

$r_1 \succeq r_2$ gilt nur, falls alle Privilegien von r_2 auch zu r_1 gehören und alle Benutzer von r_1 auch Benutzer von r_2 sind.

$$\begin{aligned} r_1 \succeq r_2 \quad \Rightarrow \quad & rollen\_priv(r_2) \subseteq rollen\_priv(r_1) \wedge \\ & rollen\_benutzer(r_1) \subseteq rollen\_benutzer(r_2) \end{aligned}$$

Die Funktion *rollen_benutzer* und *rollen_priv* müssen hierbei zur Unterstützung der Rollenhierarchien neu definiert werden. *rollen_benutzer* gibt alle Benutzer von r sowie alle Benutzer der Kindrollen von r zurück.

$$rollen\_benutzer(r) = \{b \mid b \in BENUTZER \wedge r' \in \{kindrollen(r) \cup r\} \wedge (b, r') \in BZ\}$$

rollen_priv ermittelt alle Privilegien der Rolle r, sowie alle Privilegien die Elternrollen von r besitzen.

$$rollen\_priv(r) = \{p \mid p \in PRIV \wedge r' \in \{elternrollen(r) \cup r\} \wedge (r', p) \in PZ\}$$

Kind- resp. Elternrollen einer Rolle ermitteln die Funktion *kindrollen* und *elternrollen*.

$$kindrollen(r \in ROLLEN) \rightarrow \{ROLLEN\}$$
$$kinderollen(r) = \{r' \mid r' \in ROLLEN \wedge (r' \rightarrow r \in AP \vee r' \rightarrow^* r \in AP)\}$$

$$elternrollen(r \in ROLLEN) \rightarrow \{ROLLEN\}$$
$$elternrollen(r) = \{r' \mid r' \in ROLLEN \wedge (r \rightarrow r' \in AP \vee r \rightarrow^* r' \in AP)\}$$

Da PACS RBAC mit DAC kombiniert, ist auch eine direkte Zuweisung der Privilegien zu Benutzern möglich. Die Menge der Benutzerprivilegien ist entsprechend $PB \subseteq BENUTZER \times PRIV$. Die einem Benutzer zugeordneten Privilegien ermittelt die folgende Funktion.

$$benutzer\_priv(b \in BENUTZER) \rightarrow \{PRIV\}$$
$$benutzer\_priv(b) = \{p \mid p \in PRIV \wedge (b, p) \in PB\}$$

Ob eine Anfrage eines Subjekts autorisiert ist, lässt sich mit der gleichnamigen Funktion *autorisiert* überprüfen.

$$autorisiert(s \in SUBJEKTE, p \in PRIV) \rightarrow \{w, f\}$$

$autorisiert(s, a, lb, o)$ ergibt w (wahr), falls das Subjekt s autorisiert ist, Aktion a mit lokalem Benutzer lb auf Objekt o auszuführen, ansonsten entsprechend f (falsch). Die Funktion lässt sich auf zwei Subfunktionen aufteilen. Die Funktion *autorisierte_rolle* überprüft, ob das Subjekt durch eine aktivierte Rolle autorisiert ist.

$$autorisierte\_rolle(s \in SUBJEKTE, p \in PRIV) \rightarrow \{w, f\}$$

$$\begin{aligned} autorisierte\_rolle(s, a, lb, o) \quad \Rightarrow \quad & \exists r \in ROLLEN \wedge \exists p \in PRIV \wedge \\ & r \in subjekt\_rollen(s) \wedge p \in rollen\_priv(r) \wedge \\ & a(p) = a \wedge lb(p) = lb \wedge o(p) = o \end{aligned}$$

Die zweite Subfunktion überprüft die einem Benutzer direkt zugewiesenen Privilegien.

$$autorisiert\_benutzer(s \in SUBJEKTE, p \in PRIV) \rightarrow \{w, f\}$$

$$\begin{aligned} autorisiert\_benutzer(s, a, lb, o) \quad \Rightarrow \quad & \exists p \in benutzer\_priv(subjekt\_benutzer(s)) \wedge \\ & a(p) = a \wedge lb(p) = lb \wedge o(p) = o \end{aligned}$$

Die Funktion zur Autorisierung ist definiert als:

$$autorisiert(s, a, lb, o) \Rightarrow autorisiert\_benutzer(s, a, lb, o) \vee autorisiert\_rolle(s, a, lb, o)$$

3.3.2.2 Administrative Delegation

Administrative Delegation bezeichnet die Abtretung der Verwaltung der Privilegien an andere Benutzer. Sie beinhaltet mehrere Teilaspekte. Zunächst ist hier die Weitergabe einzelner Privilegien zu nennen. Diese Berechtigungen können auch durch Rollen ausgeübt werden, d.h. Rollen können Privilegien besitzen, die eine administrative Delegation jener Privilegien erlauben. Die Vergabe der Privilegien geschieht immer zwischen konkreten Benutzern. Deshalb wird niemals eine Rolle als Grantor eines Privilegs auftreten, sondern immer der Benutzer, der sich zur Vergabe des Privilegs einer Rolle bedient. Die Zuweisung von Privilegien an Rollen kann von jedem Benutzer vorgenommen werden, der über entsprechende administrative Privilegien verfügt. Hier ergibt sich kein Unterschied zur direkten Zuweisung eines Privilegs an einen Benutzer. In unserem Beispielszenario aus Kapitel 3.1.4 kann so Benutzerin $ula@uzh$ die *read* Berechtigung an Objekt $ethz.o2$ an andere Benutzer weitergeben, da sie der Rolle $r1$ angehört, die dieses administrative Privileg besitzt.

Ebenfalls kann die Benutzerzuordnung in RBAC delegiert werden. Hierzu ist das spezielle administrative Privileg *Benutzerzuordnung* nötig, das Bestandteil der administrativen Privilegien einer Export Policy ist. Dieses administrative Benutzerzuordnungsprivileg ist wiederum durch administrative Delegation an andere Benutzer übertragbar. Hierbei ist zu beachten, dass ein solches Privileg nicht an eine Rolle übertragen werden kann. Die Weitergabe eines solch weitreichenden Privilegs an eine große Menge von Benutzern ist nicht sinnvoll. Eine Gruppierung der administrativen Benutzerzuordnungsprivilegien in einer Rolle kann zwar gewünscht sein, erscheint aber aufgrund der zu erwartenden kleinen Menge an Benutzern, denen eine solche Rolle zugewiesen wird, als zu aufwändig. Auch sollte die Trennung zwischen normalen Privilegien und Benutzerzuordnungsprivilegien gewahrt bleiben. Sobald die Übertragbarkeit von Benutzerzuordnungsprivilegien an eine Rolle erlaubt wird, ist aber eine Kombination denkbar. So ist es dann möglich, dass die Zugehörigkeit zu einer Rolle gleichzeitig erlaubt, über deren Mitglieder zu entscheiden. Aus diesen Gründen wurde die Vergabe von Benutzerzuordnungsprivilegien auf DAC beschränkt. Ein Beispiel dieses administrativen Benutzerzuordnungsprivilegs ist im Beispielszenario dargestellt: Benutzerin $ula@uzh$ erhält dieses Recht dort von Benutzer $ernst@ethz$ zugeteilt.

Administrative Delegation von Privilegien Für das Gewähren von Privilegien an andere Benutzer wird die Notation $\mapsto$ verwendet. $b_1 \mapsto b_2$ bedeutet entsprechend, dass Benutzer b_1 Benutzer b_2 ein Privileg gewährt hat. Hierdurch wird eine Grantbeziehung zwischen dem Grantor b_1 und dem Grantee b_2 (Benutzer, der das Privileg erhalten hat) gekennzeichnet. Eine solch Grantbeziehung kann auch transitiv sein, die dann mit $\mapsto^+$ darstellt wird. $b_1 \mapsto^+ b_2$ ist entsprechend $b_1 \mapsto q_1 \dots q_n \mapsto b_2$ für $n \geq 1$. Alle Grantors dieser Delegationskette mit Ausnahme von q_n müssen hierbei das Privileg mit Grantoption weitergegeben haben.

In PACS bilden Export Policies die Basis der globalen Zugriffskontrolle. Der Ursprung einer Delegationskette des globalen Zugriffskontrollmodells ist deshalb ein administratives Privileg einer Export Policy. Ein Benutzer kann folglich auf zwei Arten zur Weitergabe eine Privilegs berechtigt sein:

1. Der Benutzer ist vom selben Peer wie das Datenobjekt. In diesem Fall ist er zur Weitergabe des Privilegs berechtigt, falls er für das Privileg entsprechende administrative Privilegien in der Export Policy des Peers besitzt.

2. Der Benutzer besitzt ein globales administratives Privileg, das ihn zur Weitergabe des

Privilegs berechtigt.

Die Überprüfung von Punkt eins erfolgt durch die Auswertung der XACML Export Policy bzw. der darin enthaltenen administrativen Berechtigungen. Die Evaluation der Export Policy wurde in Kapitel 2.4.4.1 beschrieben und ist nicht Gegenstand dieser Darstellung. Die Funktion

$$lokaler\_benutzer\_objekt(o \in OBJECT, lb \in BENUTZER) \rightarrow \{w, f\}$$

ermittelt, ob Benutzer b den selben Peer als Ursprung hat wie Objekt o. $grant\_priv\_lokal(a \in AKTION, lb \in BENUTZER, o \in OBJEKTE) \rightarrow \{w, f\}$ führt die Export Policy Evaluation durch. Sie liefert w zurück, wenn $lb \in BENUTZER$ ein lokaler Benutzer des Peers ist und für Objekt $o \in OBJEKTE$ zudem das administrative Privileg besitzt, das Privileg mit Aktion $a \in AKTION$ weiterzugeben. Es ist hierbei unerheblich, ob das Privileg an einen Benutzer oder an eine Rolle weitergegeben werden soll. Ebenso ist es irrelevant, ob das administrative Privileg zu einer Rolle des Benutzers gehört oder diesem direkt zugewiesen wurde.

Punkt zwei kann durch die Funktion $grant\_priv$ überprüft werden.

$$\begin{aligned} grant\_priv(\quad & a \in AKTIONEN, lb \in BENUTZER, \\ & o \in OBJEKTE, b \in BENUTZER) \quad \rightarrow \{w, f\} \end{aligned}$$

$$\begin{aligned} grant\_priv(a, lb, o, b) \quad = \quad & (\neg lokaler\_benutzer\_objekt(o, b) \wedge \\ & (\exists p \in (benutzer\_priv(b) \cup rollen\_priv\_benutzer(b)) \wedge \\ & a(p) = a \wedge lb(p) = lb \wedge o(p) = o \wedge go(p) = w) \vee \\ & (lokaler\_benutzer\_objekt(o, b) \wedge \\ & grant\_priv\_lokal(a, b, o) \wedge lb = b)) \end{aligned}$$

Die RBAC-Privilegien des Benutzers werden durch die Funktion $rollen\_priv\_benutzer$ ermittelt.

$$\begin{aligned} & rollen\_priv\_benutzer(b \in BENUTZER) \rightarrow \{PRIV\} \\ & rollen\_priv\_benutzer(b) \subseteq \{p \in PRIV \mid r \in benutzer\_rollen(b) \wedge (r, p) \in PZ\} \end{aligned}$$

Da Privilegien transitiv gewährt werden können, ist zudem eine Überprüfung des Delegationspfades nötig. Hierzu dient der Aufbau des Delegationsgraphen DG der nun beschrieben wird. Ein Zugriffsrecht ist die Verbindung eines Benutzers oder einer Rolle zu einem Privileg. Entsprechend ist die Menge der Zugriffsrechte durch $PZ \cup PB$ definiert. Die Zugriffsrechte eines Benutzers b ermittelt die Funktion $zugr\_benutzer$.

$$\begin{aligned} & zugr\_benutzer(b \in BENUTZER) \rightarrow \{BENUTZER \times PRIV\} \\ & zugr\_benutzer(b) = \{(b, p) \mid p \in rollen\_priv\_benutzer(b)\} \cup \{(b, p) \mid (b, p) \in PB\} \end{aligned}$$

Die Menge aller Zugriffsrechte aller Benutzer, die diese durch Rollen zugeordnet bekommen, ist definiert als:

$$\begin{aligned} ABGP \quad & \subseteq \quad BENUTZER \times PRIV \\ ABGP \quad & = \quad \bigcup_{b \in BENUTZER} zugr\_benutzer(b) \end{aligned}$$

Der Delegationsgraph wird stets für ein spezielles Privileg aufgebaut, wobei die Knoten des Delegationsgraphen die Benutzer darstellen, die ein entsprechendes Privileg besitzen. Eine gerichtete Kante $b_1 \mapsto b_2$ bezeichnet die Weitergabe eines Privilegs von Benutzer b_1 an b_2. Der Delegationsgraph ist

$$DG_p(BENUTZER, GRANTS)$$

Eine Kante im Delegationsgraph DG_{priv} für das Privileg $priv \in PRIV$ ist wie folgt definiert:

$$\begin{aligned} b_i \mapsto b_j \quad \Rightarrow \quad & \exists b \exists p((b,p) \in zugr\_benutzer(b_j) \wedge \\ & g(p) = b_i \wedge a(p) = a(priv) \wedge lb(p) = lb(priv) \wedge o(p) = o(priv)) \end{aligned}$$

Die Menge $GRANTS$ ist die Menge aller Kanten $\{b_i \mapsto b_j\}$. Das weitergegebene Zugriffsrecht wird durch die Funktion $zugr\_re$ bestimmt.

$$zugr\_re(dg \in GRANTS, a \in AKTION, lb \in BENUTZER, o \in OBJEKT) \rightarrow \{BENUTZER \times PRIV\}$$

$$\begin{aligned} zugr\_re(b_i \mapsto b_j, p) \quad = \quad & \{(b_j, p_j) \in zugr\_benutzer(b_j), (b_i, p_i) \in zugr\_benutzer(b_i) \mid \\ & a(p_j) = a(p_i) = a \wedge lb(p_j) = lb(p_i) = lb \wedge o(p_j) = o(p_i) = o \wedge \\ & g(p_j) = b_i\} \end{aligned}$$

Aus dem Delegationsgraph DG_{priv} können Delegationsketten extrahiert werden. Eine Delegationskette DK_{priv} wird auch Delegationspfad genannt und entspricht einer geordneten Sequenz von Privilegienweitergaben, wobei $a = a(priv)$, $lb = lb(priv)$ und $o = o(priv)$.

$$\begin{aligned} \langle b_1, b_2, \ldots, b_k \rangle \in DK_{priv} \quad \Leftrightarrow \quad & \forall i \in \{1, \ldots, k-1\} \wedge \forall j \in \{1, \ldots, k-2\} \wedge b_i \mapsto b_{i+1} \wedge \\ & \exists zr \exists zr'(zr \in zugr\_re((b_j \mapsto b_{j+1}), a, lb, o) \wedge \\ & zr' \in zugr\_re((b_{j+1} \mapsto b_{j+2}), a, lb, o) \wedge \\ & ts(zr) < ts(zr') \wedge go(zr) = w) \end{aligned}$$

Die Funktion $check\_zugr$ überprüft die Gültigkeit eines Zugriffsrechts. Dies geschieht durch die Überprüfung seines Delegationspfades.

$$check\_zugr(zr \in ABGP) \rightarrow \{w, f\}$$

$$check\_zugr(zr) \quad = \quad \exists(\langle lb(zr), \ldots, b(zr) \rangle \in DK_{zr})$$

Administrative Delegation von Benutzerzuordnungen Für die Zuweisung von Rollen an Benutzer wird die Notation $\hookrightarrow$ verwendet. $b_1 \hookrightarrow b_2$ gilt, falls der Grantor b_1 dem Grantee b_2 eine bestimmte Rolle zugewiesen hat. Wie bei den Privilegien kann eine solche Beziehung auch transitiv sein. Ein Benutzer ist für die Gewährung einer Benutzerzuordnung berechtigt, falls genau eine von zwei Bedingungen gilt:

1. Der Benutzer ist vom selben Peer wie die Rolle, zu der der Benutzer hinzugefügt werden soll. Zusätzlich besitzt der Benutzer gemäß der Export Policy des Peers das Benutzerzuordnungsprivileg für diese Rolle.

2. Der Benutzer besitzt ein Benutzerzuordnungsprivileg auf globaler Ebene für diese Rolle.

Punkt eins kann durch Auswertung der Export Policy überprüft werden. Die Funktion *lokaler_benutzer_rolle* ermittelt, ob die Rolle zum selben Peer wie der Benutzer gehört.

$$lokaler\_benutzer\_rolle(b \in BENUTZER, r \in ROLLEN) \rightarrow \{w, f\}$$

grant_rolle_lokal wertet die XACML Policy dahingehend aus, ob der Benutzer das entsprechende Benutzerzuordnungsprivileg für diese Rolle besitzt.

$$grant\_rolle\_lokal(r \in ROLLEN, b \in BENUTZER) \rightarrow \{w, f\}$$

Beide Bedingungen lassen sich durch die Funktion *grant_rolle* überprüfen.

$$grant\_rolle(r \in ROLLE, b \in BENUTZER) \rightarrow \{w, f\}$$

$$\begin{aligned} grant\_rolle(r, b) \quad = \quad & (\neg lokaler\_benutzer\_rolle(r, b) \wedge (\exists (b, r, g, w, t) \in BZ) \vee \\ & (lokaler\_benutzer\_rolle(r, b) \wedge grant\_rolle\_lokal(r, b)) \end{aligned}$$

Auch für Benutzerzuordnungen einer bestimmten Rolle kann ein Delegationsgraph erstellt werden. Dieser ist definiert als

$$DG_{bz}(BENUTZER, GRANTS)$$

Eine Kante im Delegationsgraph DG_{bz} für die Rolle r ist wie folgt definiert:

$$b_i \hookrightarrow b_j \Rightarrow \exists bz(bz \in BZ \wedge g(bz) = b_i \wedge b(bz) = b_j \wedge r(bz) = r)$$

Die Menge $GRANTS$ entspricht wiederum der Menge aller Kanten $\{b_i \hookrightarrow b_j\}$. Das weitere Vorgehen erfolgt analog zur administrativen Delegation von Privilegien. Eine Delegationskette DK_{bz} ist hierbei eine geordnete Sequenz von Weitergaben administrativer Benutzerzuordnungsprivilegien für die Rolle r.

$$\begin{aligned} \langle b_1, b_2, \ldots, b_k \rangle \in DK_{bz} \quad \Leftrightarrow \quad & \forall i \in \{1, \ldots, k-1\} \wedge \forall j \in \{1, \ldots, k-2\} \wedge b_i \hookrightarrow b_{i+1} \wedge \\ & \exists bz \exists bz' (bz \in benutzer\_zu((b_j \hookrightarrow b_{j+1}), r) \wedge \\ & bz' \in benutzer\_zu((b_{j+1} \hookrightarrow b_{j+2}), r) \wedge \\ & ts(bz) < ts(bz') \wedge go(bz) = w) \end{aligned}$$

Die Funktion *benutzer_zu* bestimmt hierbei das weitergegebene Benutzerzuordnungsprivileg.

$$\begin{aligned} benutzer\_zu(dg \in GRANTS, r \in ROLLEN) \quad & \rightarrow \quad \{BZ\} \\ benutzer\_zu(b_i \hookrightarrow b_j, r) \quad & = \quad \{(b_j, r, b_i, go, ts) \in BZ\} \end{aligned}$$

Die Funktion *check_bz* überprüft die Gültigkeit einer Benutzerzuordnung. Dies geschieht durch die Überprüfung deren Delegationspfades.

$$check\_bz(bz \in BZ) \rightarrow \{w, f\}$$
$$check\_bz(bz) = \exists lb(lb \in lokale\_grants(bz) \wedge \langle lb, \ldots, b(bz) \rangle \in DK_{bz})$$

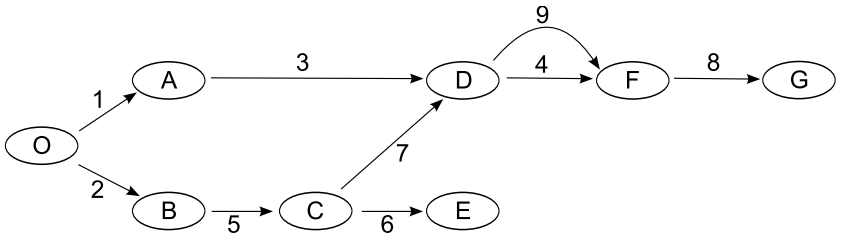


Abbildung 3.10: Delegationsgraph vor dem Widerruf

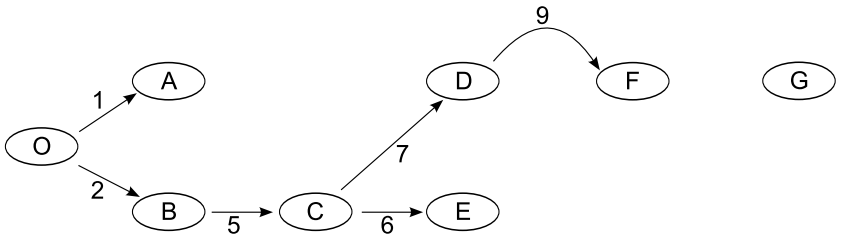


Abbildung 3.11: Delegationsgraph nach dem Widerruf von Benutzer A

3.3.2.3 Kaskadierendes Widerrufen

Benutzer können ein erteiltes Privileg oder eine Benutzerzuordnung nur widerrufen, wenn sie von ihnen direkt erteilt wurde. PACS unterstützt hierbei das kaskadierende Widerrufen von Privilegien und Benutzerzuordnungen. Dies bedeutet, dass beim Widerruf eines Privilegs auch alle Weiterreichungen des Privilegs bzw. der Benutzerzuordnung ungültig werden. Der Widerruf eines Privilegs muss hierbei so durchgeführt werden, dass die Autorisierungen der Benutzer dem Zustand entsprechen der herrschen würde, wenn das Privileg gar nicht erteilt worden wäre. Es gibt verschiedene Varianten des kaskadierenden Widerrufs. PACS verwendet die Variante von Griffiths u.a. [GW76] mit Fagins Korrektur [Fag78]. Das Beispiel in Abbildung 3.10 dient dabei als Veranschaulichung. Die Buchstaben kennzeichnen hierbei die Benutzer. Jede Kante des Delegationsgraphen ist ein Zugriffsrecht zwischen zwei Benutzern mit einem Zeitstempel. In Abbildung 3.10 sind Zeitstempel als Zahl an Kanten geheftet. Hierdurch kann die zeitliche Abfolge der Privilegienerteilung nachvollzogen werden. Entzieht nun Benutzer A zum Zeitpunkt 10 das vergebene Privileg (A, D) werden auch jene Privilegien entzogen, die auf diesem basieren (kaskadierendes Widerrufen). Im Beispiel sind dies das Privileg mit Zeitstempel 4 (D, F) und das Privileg mit Zeitstempel 8 zwischen (F, G). Das Resultat ist in Abbildung 3.11 dargestellt. Das Privileg zwischen (F, G) muss entzogen werden, da es auf dem widerrufenen Privileg (A, D) basiert. Das zweite Erteilen des Privilegs von (D, F) zum Zeitpunkt 9 bleibt bestehen, da dieses auf dem Privileg (C, D) basiert. F könnte G das Privileg deshalb wiederum erteilen, diesmal aber basierend auf dem Privileg, das transitiv durch C gewährt wurde. Hätte statt A im Beispiel 3.10 C das Privileg (C, D) entzogen, würde zwar das Privileg (D, F) zum Zeitpunkt 9 ebenfalls widerrufen, das Privileg (F, G) bliebe allerding erhalten.

Beim Widerruf eines Privilegs muss der Delegationsgraph angepasst werden. Widerruft ein Benutzer b_i ein Privileg *priv*, das er an Benutzer b_j weitergegeben hat, müssen auch die Berechtigungen von allen Benutzern entzogen werden, an die es b_j weitergereicht hat. Da der lokale Benutzer Bestandteil des Privilegs ist, kann eine Überprüfung alternativer

Delegationsketten mit Ursprung bei einem anderen lokalen Benutzer unterbleiben. Daraus ergeben sich folgende Vorgehensweisen.

Widerruf eines Privilegs Das zu löschende Zugriffsrecht sei $(b_j, a, lb, o, g, go, ts)$.

1. Löschen des Zugriffsrechts. Ist die Grantoption $go = f$ terminiert der Algorithmus, ansonsten wird mit Punkt zwei fortgefahren.

2. Ermitteln des minimalen Zeitstempels. Hier wird zunächst untersucht, ob b_j das Zugriffsrecht noch von einem anderen Grantor zu einem anderen Zeitpunkt erhalten hat.

$$\begin{aligned} PB' &= \{(b,p) \mid (b,p) \in zugr\_benutzer(b_j) \wedge \\ &\quad a(p) = a \wedge lb(p) = lb \wedge o(p) = o \wedge go = w\} \end{aligned}$$

   Aus dieser Menge wird das Zugriffsrecht mit dem minimalen Zeitstempel min_ts ermittelt.

$$min\_ts = (b,p) \in PB' \wedge (\forall (b',p') \in PB \setminus \{p\} ts(p') > ts(p))$$

   Ist $PB' = \emptyset$, ist der minimale Zeitstempel $min\_ts = \infty$.

3. Nun werden alle zu löschenden Zugriffrechte ausgewählt, die von b_j vor dem Zeitpunkt min_ts gewährt wurden.

$$\begin{aligned} TODELETE &= \{(b,p) \mid (b,p) \in ABGP \cup PB \wedge \\ &\quad a(p) = a \wedge lb(p) = lb \wedge o(p) = o \wedge g(p) = b_j \wedge \\ &\quad ts(p) < min\_ts\} \end{aligned}$$

   $\forall z \in TODELETE$ führe den Löschalgorithmus aus (rekursiver Aufruf).

Wird ein Privileg von einer Rolle entfernt, wird das Zugriffsrecht aus PZ gelöscht. Anschließend muss die Menge aller Benutzer ermittelt werden, die dieser Rolle zugewiesen sind. Dies übernimmt die zuvor definierte Funktion *rollen_benutzer*. Für alle Benutzer dieser Menge muss das kaskadierende Widerrufen für das entzogene Privileg durchgeführt werden, um von diesem abgeleitete Privilegien ebenfalls zu löschen.

Widerrufen einer Benutzerzuordnung Beim Widerrufen einer Benutzerzuordnung zu Benutzer b_j, die von b_i gewährt wurde, ist ein kaskadierendes Widerrufen nur nötig, falls alle Delegationsketten von lokalen Benutzern lb zu b_j, b_i beinhalten. Hierzu müssen zunächst alle lokalen Benutzer mit entsprechendem Benutzerzuordnungsprivileg ermittelt werden. Dies geschieht durch die Funktion $grant\_rolle\_lokal(r \in ROLLEN, lb \in BENUTZER) \subseteq BENUTZER$.

$$DK' = \{dk \mid dk \in DK \wedge lb \in grant\_priv\_lokal(r,b) \wedge dk = \langle lb, \ldots, b_j \rangle\}$$

$$\forall dk \in DK', b_i \in dk$$

Muss gemäß dieser Überprüfung ein kaskadierender Widerruf durchgeführt werden, ist das Vorgehen wie folgt. Sei die zu löschende Benutzerzuordnung (b, r, g, go, t).

1. Löschen der Benutzerzuordnung aus BZ. Dies impliziert ein kaskadierendes Widerrufen aller der Rolle zugewiesenen Privilegien für diesen Benutzer. Ist die Grantoption der Benutzerzuordnung $go = f$ terminiert der Algorithmus, ansonsten ist mit Punkt zwei fortzufahren.

2. Ermitteln des minimalen Zeitstempels. Hierzu wird untersucht, ob Benutzer b die Rolle noch von einem anderen Grantor zu einem anderen Zeitpunkt zugewiesen bekommen hat.

$$BZ' \quad = \quad \{(b, r, g', w, t') \in BZ\}$$

   Die Ermittlung des Zugriffsrechts mit minimalen Zeitstempel min_ts aus dieser Menge geschieht wie folgt.

$$min\_ts = (b, r) \in BZ' \wedge (\forall (b', r') \in PB \setminus \{r\} ts(r') > ts(r))$$

   Ist $BZ' = \emptyset$, ist der minimale Zeitstempel $min\_ts = \infty$.

3. Nun werden alle zu löschenden Benutzerzuordnungen ausgewählt, die von b vor dem Zeitpunkt min_ts gewährt wurden.

$$TODELETE \quad = \quad \{(b, r, g', go, ts) \in BZ \mid g' = b \wedge ts < min\_ts\}$$

   $\forall bz \in TODELETE$ führe den Löschalgorithmus aus (rekursiver Aufruf).

Hierbei ist zu beachten, dass beim Löschen der Benutzerzuordnung alle der Rolle zugewiesenen Privilegien für den Benutzer ebenfalls kaskadierend gelöscht werden müssen. Da dem Benutzer die Privilegien nie direkt gewährt wurden, ist eine Löschung der Privilegien nicht nötig. Allerdings kann der Benutzer andere Privilegien, basierend auf den durch die Rolle gewährten, erteilt haben. Diese müssen entsprechend ebenfalls entzogen werden. Eine solche Löschoperation ist sehr aufwändig, insbesondere da die Privilegien und Benutzerzuordnungen in PACS verteilt gespeichert werden. Das Entziehen von Privilegien oder Rollen ist aber keine sehr häufige Aktion, weshalb auch ein hohe Aufwand toleriert werden kann.

3.3.2.4 Zeitstempel

Bislang wurde davon ausgegangen, dass die vergebenen Zeitstempel zum Aufbau des Delegationsgraphen eindeutig sind. In verteilten Systemen ist die Synchronisation der Zeit zwischen den Teilnehmern sehr aufwändig. Für den Delegationspfad ist dies aber nicht nötig, denn für Privilegien genügt ein koordinierter Zeitstempel. Zeitstempel in PACS sind nicht absolut, sondern relativ. Der Mechanismus hierzu wird im Folgenden erläutert.

Jeder Peer verfügt über einen lokalen Zähler „Grantzähler“. Bei jedem Privileg, das ein Benutzer des Peers vergibt, wird dieser Zähler inkrementiert und als Teil des Zeitstempels des Privilegs gespeichert. Dieser Zähler dokumentiert dadurch die Reihenfolge der vom Benutzer des Peers gewährten Privilegien. Für den Aufbau des Graphen wird zusätzlich noch die Information benötigt, auf welchen Privilegien die Weitergabe eines Privilegs basiert. Anders ausgedrückt, es ist erforderlich festzuhalten welche Privilegien der Benutzer zum Zeitpunkt der Weitergabe besaß. Hierzu wird der Grantzähler des Grantees zum Zeitpunkt der Vergabe des Privilegs ebenfalls gespeichert. Wenn Benutzerin *ula@uzh* mit Zähler 1 ein Privileg an Benutzer *ernst@ethz* mit Zähler 0 weitergibt, so wird als Zeitstempel für das neue

Privileg der inkrementierte Zähler von Benutzerin $ula@uzh = 2$ und der Zähler von Benutzer $ernst@ethz = 0$ gespeichert. Der Zeitstempel im vorliegenden Beispiel ist somit *(Zähler Grantor, Zähler Grantee)* $(2, 0)$.

Zwei Zugriffsrechte für Aktion a, lokalen Benutzer lb und Objekt o seien gegeben als $z_1 = zugr\_re((b_1 \mapsto b_2), a, lb, o)$ und $z_2 = zugr\_re((b_2 \mapsto b_3), a, lb, o)$. Die Ordnung der Zeitstempel $ts_1(z_1) = (g_1, b_1)$ und $ts(z_2) = (g_2, b_2)$ ist dann wie folgt definiert:

$$ts_1 < ts_2 \Leftrightarrow (g_1, b_1) < (g_2, b_2) \Leftrightarrow b_1 < g_2$$

Entsprechend gilt die Umkehrung:

$$ts_1 > ts_2 \Leftrightarrow \neg(ts_1 < t_2) \Leftrightarrow b_1 > g_2$$

Da der Grantor beim Abspeichern des Privilegs den Zähler des Grantees erfragen muss, ist dieser in den Prozess der Privilegienvergabe involviert. Hierdurch ergeben sich zwei mögliche Probleme:

1. Der Grantee bzw. der Peer des Grantees ist nicht verfügbar. In diesem Fall ist die Weitergabe des Privilegs nicht möglich. Als Alternative kann der Grantorzähler aber auch durch Abfrage der durch den Grantee vergebenen Rechte ermittelt werden.

2. Der Grantee übermittelt dem Grantor einen falschen Zählerstand. Hierbei lassen sich zwei Fälle unterscheiden:

   (a) Der vom Grantee erhaltene Zählerstand ist zu klein. Dies verändert die Grundlage der vom Grantee schon weitergegebenen Rechte. Dieses Fehlverhalten des Grantees betrifft nur dessen Einflussbereich (eben jene von Grantee weitergegebenen Rechte). Deshalb ist dieses Fehlverhalten für den Grantor nicht kritisch. Der Grantor muss lediglich sicherstellen, dass ein Privileg, das er zu einem späteren Zeitpunkt an den Grantee gewährt, auch einen höheren oder gleich großen Grantee Zähler besitzt als das vorhergehende Privileg. Zudem kann die Korrektheit des Zeitstempels durch die Überprüfung der schon von Grantee vergebenen Privilegien überprüft werden.

   (b) Der vom Grantee erhaltene Zählerstand ist zu groß. In diesem Fall wird der Zeitpunkt des Erhalts des Privilegs in die Zukunft verlagert, was zum Nachteil des Grantees ist.

Die vorgestellten möglichen Fehler sind für die Privilegienvergabe als Ganzes nicht kritisch, da die Auswirkungen dieser Fehler lokal beschränkt bleiben.

Der von Abbildung 3.10 bekannte Delegationsgraph ist in Abbildung 3.12 nochmals mit Zeitstempeln bestehend aus Zählern zu sehen. Die Zahlen an den Kanten entsprechen hier *(Zähler Grantor, Zähler Grantee)*.

Das hier vorgestellte globale Zugriffskontrollmodell erfüllte durch die Kombination von RBAC und DAC mit administrativer Delegation die gestellten Anforderungen. Die in diesem Kapitel vorgestellten Regeln definieren diese Kombination zwischen RBAC und DAC exakt und spezifizieren so das globale Zugriffskontrollmodell von PACS. PACS nutzt die hier vorgestellten Funktionen zum Zugriff auf das Zugriffskontrollmodell bei der Verwaltung und Durchsetzung der Privilegien.

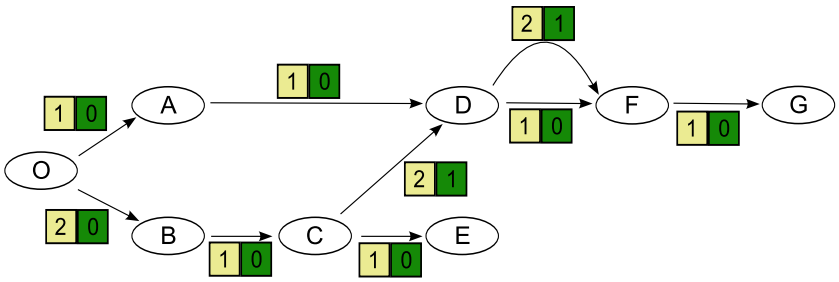


Abbildung 3.12: Delegationsgraph mit relativen Zeitstempeln

3.4 Verwaltung von Privilegien

Grundlage für die Verwaltung von Privilegien ist das vorgestellte Zugriffskontrollmodell von PACS. Durch die Privilegienverwaltung können gemäß diesem Modell Zugriffsrechte und Benutzerzuordnungen von den Benutzern gewährt und widerrufen werden. Sie bildet sozusagen die Benutzerschnittstelle zum Zugriffsmodell. Die Privilegienverwaltung garantiert hierbei anhand des Zugriffskontrollmodells die Korrektheit der von ihr verwalteten Privilegien.

Grundsätzlich kann die Überprüfung der Privilegien und Benutzerzuordnungen bei jeder Anfrage oder bei der Änderung der Privilegien erfolgen. Da es wesentlich mehr Anfragen gibt als Änderungen an den Privilegien ist es effizienter, die Überprüfung der Privilegien bei Änderungen durchzuführen. Bei Abfragen ist dann die Korrektheit der verwalteten Benutzerzuordnungen und Zugriffsrechte garantiert.

Auch bei der Privilegienverwaltung gilt es, die lokale und globale Ebene zu unterscheiden.

- Die lokale Ebene kümmert sich um Veränderungen an der Export Policy des Peers und veranlasst die durch Veränderungen der Export Policy nötigen Änderungen an globalen Privilegien. Änderungen die globale Privilegien beeinflussen sind:
  - Einfügen neuer Regeln in XACML Policies
  - Hinzufügen neuer Elemente zu XACML-Regeln
  - Löschen von Elementen von XACML-Regeln
  - Löschen von XACML-Regeln
  - Löschen von XACML Policies
- Die globale Ebene behandelt das Gewähren und Entziehen von globalen Privilegien, sowie die dadurch resultierenden Auswirkungen auf andere Privilegien.

3.4.1 Gewähren neuer globaler Privilegien

Gewährt ein Benutzer einem anderen Benutzer oder einer Rolle ein globales Privileg, überprüft die Privilegienverwaltung vor dem Speichern der neuen Zuordnung im Privilegienspeicher die Korrektheit der neuen Zuordnung.

Die Privilegienverwaltung kontrolliert, ob der Grantor die nötigen administrativen Privilegien zur Vergabe des Privilegs besitzt. Dies wird durch Auswertung der Funktion *grant_priv* überprüft. Als Optimierung kann der Grantor alle Privilegien, die ihn zur Vergabe des Privilegs berechtigen, an die Privilegienverwaltung übermitteln. Die Privilegienverwaltung muss

dann nicht nach den relevanten Privilegien suchen, sondern prüft lediglich die Existenz und Aktualität der vom Grantor gelieferten Privilegien. Dies geschieht durch Abgleich mit dem Inhalt des Privilegienspeichers. Um den Besitz des nötigen administrativen Privilegs zu prüfen, wird die schon erläuterte Funktion *grant_priv* verwendet. Wurden Privilegien vom Grantor bei der Anfrage bereitgestellt, können die Mengen BZ, PZ und PB auf die vom Grantor gelieferten Privilegien beschränkt werden. Ist die Überprüfung erfolgreich, ist die Vergabe des Privilegs autorisiert und die Privilegienverwaltung veranlasst die Speicherung des Privilegs im Privilegienspeicher.

3.4.2 Widerrufen globaler Privilegien

Auch vor dem Löschen eines Privilegs muss der Anfragende seine Berechtigung für diesen Löschvorgang vor der Privilegienverwaltung nachweisen. Da jeder Grantor nur die von ihm direkt vergebenen Privilegien löschen darf, ist zu prüfen, ob das betreffende Privileg vom Anfragenden gewährt wurde. Nach erfolgreicher Prüfung dieser Frage wird das kaskadierende Widerrufen dieses Privilegs von der Privilegienverwaltung initiiert.

3.4.3 Gewähren neuer globaler Benutzerzuordnungen

Weist ein Benutzer einem anderen Benutzer eine neue Rolle zu, muss die Autorisierung des Grantors für diese Zuweisung durch die Privilegienverwaltung überprüft werden. Der Grantor kann als Optimierung hierbei seine relevanten Benutzerzuordnungsprivilegien an die Privilegienverwaltung liefern. Ist dies der Fall, müssen diese lediglich auf ihre Aktualität überprüft werden. Sonst muss die Privilegienverwaltung entsprechende Benutzerzuordnungsprivilegien des Benutzers selbst suchen. Mit diesen gefundenen bzw. überprüften Benutzerzuordnungsprivilegien erfolgt die Auswertung der Funktion *grant_rolle*. Nur bei erfolgreicher Prüfung wird die neue Benutzerzuordnung von der Privilegienverwaltung akzeptiert und im Privilegienspeicher gespeichert.

3.4.4 Widerrufen globaler Benutzerzuordnungen

Vor dem Löschen einer Benutzerzuordnung wird überprüft, ob diese auch vom anfragenden Benutzer gewährt wurde, denn nur er ist berechtigt, diese wieder zurückzunehmen. Ist dies der Fall, wird das kaskadierende Widerrufen dieser Benutzerzuordnung durchgeführt.

3.4.5 Veränderungen an Export Policies

Bestimmte Änderungen an Export Policies haben Auswirkungen auf globale Privilegien. Es haben nur Regeln der XACML Policy Auswirkungen auf globale Privilegien, die lokalen Benutzern administrative Privilegien gewährt haben. Änderungen an Subjekt, Objekten, Aktionen und administrativen Attributen dieser Regeln beeinflussen globalen Privilegien direkt.

Das PACS-Modul des Peers wacht lokal über Veränderungen der Export Policy. Nur einer definierten Gruppe von Benutzern, z.B. Systemadministratoren, ist es gestattet, Änderungen an dieser Export Policy vorzunehmen. Während also Aufforderungen zum Gewähren neuer Privilegien von externen Benutzern und Peers erfolgen können, werden Veränderungen an der Export Policy immer lokal initiiert. Die Änderungen lassen sich untergliedern in Löschung und Erzeugung von Policies, Regeln oder deren Bestandteile.

3.4.5.1 Einfügen neuer XACML-Regeln

Die Erzeugung neuer XACML-Regeln in der Export Policy hat keine Auswirkungen auf schon vergebene globale Privilegien. Durch die Erzeugung einer neuen Regel wird ein weiteres Privileg erzeugt, das anschließend durch administrative Delegation an andere Benutzer weitergegeben werden kann. Schon erteilte globale Privilegien müssen deshalb nicht angepasst werden.

3.4.5.2 Hinzufügen neuer Elementen zu einer XACML-Regel

Bei dieser Modifikation von Teilen einer Regel ist derselbe Sachverhalt wie bei der Erzeugung einer neuen XACML-Regel gegeben. Denn auch bei Hinzunahme eines neuen Subjekts, einer neuen Aktion, eines neuen Objekts oder eines neuen Grantors wird lediglich ein neues Privileg auf lokaler Ebene geschaffen und kein bestehendes globales Privileg widerrufen.

3.4.5.3 Löschen eines Elements aus einer XACML-Regel

Wird ein Element einer Regel (Subjekt, Aktion, Objekt oder Grantor) entfernt oder die Grantoption widerrufen, wird globalen Privilegien oder Benutzerzuordnungen, die auf dieser Regel aufbauenden die Grundlage entzogen. Deshalb müssen betroffene globale Privilegien und Benutzerzuordnungen widerrufen werden. Beim Löschen eines Elements aus einer XACML-Regel wird zwischen XACML Regeln, die Privilegien entsprechen (im Folgenden Privilegienregeln genannt), und XACML-Regeln der „Role Assignment“ Policies (im Folgenden Zuordnungsregeln genannt) unterschieden.

Während bei Privilegienregeln globale Privilegien betroffen sind, sind bei Zuordnungsregeln auch noch Benutzerzuordnungen betroffen. Zunächst werden Privilegienregeln betrachtet.

Löschen von Elementen einer Privilegienregel Der Prozess zum Löschen betroffener globaler Privilegien ist in drei Schritte unterteilt. Hier gilt es, je nach Änderung folgende Fälle zu unterscheiden:

- Eine Aktion oder ein Objekt wurde gelöscht. Hierdurch müssen alle globalen Privilegien, die diese Aktion bzw. dieses Objekt beinhalten, gelöscht werden.
- Ein Grantor wurde entfernt, bzw. die Grantoption wurde widerrufen. In diesem Fall sind die globalen Privilegien, die diesen Grantor besitzen, zu widerrufen.

Die Grantees der so betroffenen globalen Privilegien können ermittelt werden, da bekannt ist, dass jeweils ein lokaler Benutzer des Peers Grantor sein muss. Jeder Peer protokolliert (schon um eine lokale Sicherung zu besitzen) alle von seinen Benutzern vergebenen Privilegien und Benutzerzuordnungen. In dieser lokalen Sicherung wird nach den zu widerrufenden globalen Privilegien gesucht. Jedes Element der so ermittelten Menge betroffener globaler Privilegien wird durch die Privilegienverwaltung widerrufen (siehe Kapitel 3.4.2). Hierbei ist jedoch eine Besonderheit zu beachten. Wie zuvor ausgeführt, darf nur der Grantor das Privileg bzw. die Benutzerzuordnung widerrufen. In diesem Fall geschieht der Aufruf zum kaskadierenden Entziehen allerdings nicht vom Grantor. Um dennoch eine Löschung der globalen Privilegien zu erreichen, muss der Anfragende dem Privilegienspeicher die neue modifizierte Export Policy des Peers präsentieren und so darlegen, dass die Grundlage für das globale Privileg bzw. die Benutzerzuordnung, die gelöscht werden soll, nicht mehr existiert.

Löschen von Elementen einer Zuordnungsregel Beim Löschen von Elementen einer Zuordnungsregel wurde einem oder mehreren Benutzern eine Rolle oder das Benutzerzuordnungsprivileg entzogen. In beiden Fällen hat dies Auswirkungen auf globale Privilegien.

Wurde eine Rolle entzogen, müssen für die betroffenen Benutzer die administrativen Privilegien dieser Rolle (lokale und globale) widerrufen werden. Die lokalen administrativen Privilegien, die einer Rolle zugewiesen wurden, ergeben sich aus der Export Policy. Für diese wird der Prozess zum Löschen lokaler Privilegienregeln für die administrativen Privilegien der lokalen Ebene durchgeführt. Die globalen Privilegien, die einer Rolle zugewiesen wurden, müssen für diese Benutzer widerrufen werden.

Wurden administrative Privilegien widerrufen, ist darüber hinaus die Benutzerzuordnung im globalen Privilegienspeicher betroffen. In diesem Fall müssen alle auf der Basis des widerrufenen Privilegs gemachten Benutzerzuordnungen auf der globalen Ebene widerrufen werden.

Der Prozess zum Löschen betroffener globaler Benutzerzuordnungen gliedert sich wie folgt. Zunächst muss die Menge betroffener globaler Benutzerzuordnungen bestimmt werden. Rolle und Grantor der zu widerrufenden globalen Benutzerzuordnungen ergeben sich aus der modifizierten Zuordnungsregel. Mit diesen Angaben können in der lokalen Sicherung alle von lokalen Benutzern aufgrund eines administrativen Privilegs gemachten Benutzerzuordnungen identifiziert werden. Anschließend wird für jede globale Benutzerzuordnung dieser Menge der Widerruf derselben eingeleitet (siehe Kapitel 3.4.4). Zur Autorisierung für diese Löschung gegenüber der Privilegienverwaltung dient wiederum die neue modifizierte Export Policy.

3.4.5.4 Löschen einer XACML-Regel

Beim Löschen einer XACML-Regel sind die Auswirkungen noch größer als beim Löschen eines Elements einer XACML-Regel. Handelt es sich um eine Privilegienregel, muss die Menge der Benutzer bestimmt werden, die die durch diese Regel definierte Menge an administrativen Privilegien verliert. Für diese Menge an Grantors und Privilegien wird dann der Prozess zum Löschen von Elementen einer Privilegienregel durchgeführt.

Handelt es sich um eine Regel einer Rule Assignment Policy, muss die Menge der Benutzer ermittelt werden, die nun nicht mehr Mitglied der Rolle sind. Für diese Benutzer müssen alle globalen Privilegien, die sie aufgrund ihrer Rollenzugehörigkeit erteilt haben, entzogen werden. Für jeden dieser Benutzer wird entsprechend das Vorgehen beim Löschen eines Elements einer Zuordnungsregel wiederholt.

3.4.5.5 Löschen ganzer XACML Policies

Für das Löschen ganzer Policies wird der Löschprozess für einzelne XACML Regeln rekursiv durchgeführt, bis die Policy keine Regel mehr enthält. Ist dies erreicht, wurden alle Verbindungen der XACML Policy zur globalen Ebene gelöscht. So kann auch die verbliebene XACML Policy gelöscht werden.

3.5 Sicheres verteiltes Speichern von Privilegien

Bei den vorhergehenden Ausführungen war immer allgemein von einem Privilegienspeicher die Rede, wenn es um die Speicherung globaler Privilegien ging. Dieses Kapitel betrachtet diesen Privilegienspeicher detaillierter und beschreibt die Techniken, die für den Aufbau eines solchen Speichers in einem P2P-Umfeld ohne globale Koordination nötig sind. Wie schon in

Kapitel 2.9 erläutert, stellt die verlässliche Speicherung von Daten in einem P2P-Netzwerk hohe Anforderungen. Insbesondere durch die sich ständig verändernde Menge der Teilnehmer und die potentiell vorhandenen bösartigen Peers im Netzwerk wird eine verlässliche Speicherung erschwert. Um dies dennoch zu garantieren, ist ein erheblicher Mehraufwand zu leisten, der in den Kapiteln 3.5.5, 3.5.6 und 3.5.7 genauer beschrieben wird. Die verlässliche Speicherung garantiert hierbei drei Grundeigenschaften:

Dauerhafte Speicherung Im Privilegienspeicher abgelegte Datenobjekte sind darin dauerhaft gespeichert.

Verfügbarkeit Die im Privilegienspeicher abgelegten Datenobjekte können jederzeit mit sehr großer Wahrscheinlichkeit aufgefunden werden und dies unabhängig davon, wo sich der Anfragende bzw. die Daten im Netzwerk befinden.

Gewährleistete Datenintegrität Der Privilegienspeicher erkennt und filtert ungültige Privilegien und Daten bösartiger Peers.

In diesem Kapitel wird die Unterscheidung zwischen Benutzer und Peer weitestgehend aufgehoben. Grund hierfür ist neben der übersichtlicheren Darstellung vor allem, dass die Speicherung der Privilegien immer im Verantwortungsbereich des Peers liegt. Ein Benutzer gehört zu einem Peer und wird durch diesen repräsentiert. Der Peer des Benutzers kümmert sich deshalb stellvertretend für ihn um die Speicherung und das Abfragen der im Privilegienspeicher gespeicherten Daten.

3.5.1 Speichern der Informationen des Zugriffskontrollmodells

Die Informationen des Zugriffskontrollmodells müssen auf geeignete Art und Weise im Privilegienspeicher gespeichert werden. Die globalen Privilegien werden in Form zweier nicht normalisierter Relationen gespeichert, wie vom Autor in [SDZ08] bereits grundsätzlich erläutert. Es handelt sich hierbei um die Relationen Zugriffsrecht und Benutzerzuordnung mit den dazugehörigen Attributen. Unterstrichene Attribute kennzeichnen hierbei den Primärschlüssel.

Zugriffsrecht mit den Attributen *Grantee* (das den Benutzer oder die Rolle enthält, dem das Recht zugeteilt wurde), *Datenobjekt*, *Aktion*, *Lokaler Benutzer*, *Grantoption*, *Grantor*, *Zähler Grantor*, *Zähler Grantee* und *Rolle* (zeigt an, ob es sich beim Grantee um eine Rolle handelt).

Benutzerzuordnung mit den Attributen *Benutzer*, *Rolle*, *Grantor*, *Grantoption*, *Zähler Grantor* und *Zähler Grantee*.

Eine Rollenhierarchie kann lediglich auf lokaler Ebene definiert werden und muss folglich nicht auf der globalen Ebene gespeichert werden. Je nach Art der Durchsetzung, die für PACS gewählt wurde, unterscheiden sich Art und Umfang der Privilegien und Benutzerzuordnungen, die im Privilegienspeicher abgelegt werden. Neben Privilegien und Benutzerzuordnungen können auch andere Daten, die für die Durchsetzung der Privilegien nötig sind, im Privilegienspeicher abgelegt werden.

Die Informationen, die im Privilegienspeicher abgelegt werden, sind grundsätzlich für jeden Teilnehmer des Netzwerks einsehbar. Dies ist ein Sicherheitsproblem, da so jeder Teilnehmer erfahren kann, welcher Benutzer welche Privilegien und Rollen besitzt. Dieses Problem lässt

sich durch Verschlüsselung der zu speichernden Information und entsprechender Verteilung der Schlüssel beheben (siehe Sturm u.a. [SDZ08]). Allerdings gestaltet sich dann die Verwaltung der gespeicherten Informationen wesentlich aufwändiger. In Abwägung der beiden Punkte wurde deshalb in PACS auf die Verschlüsselung der Informationen im Privilegienspeicher verzichtet.

3.5.2 Bösartige Peers

Eine Bedrohung für einen verlässlichen Privilegienspeicher sind die möglicherweise im Netzwerk vorhandenen bösartigen Peers.

Ein *gutartiger* Peer hält sich immer an das von PACS vorgeschriebene Protokoll, ansonsten handelt es sich um einen *bösartigen* Peer. Ein bösartiger Peer kann sich hierbei beliebig falsch verhalten. Er kann sich zudem mit anderen bösartigen Peers verbünden und mit diesen zusammenarbeiten. Im „Worst Case“ Szenario kooperiert er sogar mit allen im Netzwerk vorhandenen bösartigen Peers. Allerdings wird angenommen, dass auch bösartige Peers dem PACS-Protokoll folgen, solange es in ihrem eigenen Interesse und zu ihrem Vorteil ist. Das Ziel eines bösartigen Peers ist es, unautorisierten Zugriff auf gespeicherte Daten zu erhalten oder andere autorisierte Benutzer am Datenzugriff zu hindern.

Des Weiteren gilt die Annahme, dass bösartige Peers nur über beschränkte Rechnerressourcen verfügen. Deshalb ist es ihnen nicht möglich, die zugrundeliegenden Verschlüsselungsverfahren zu brechen. Hierdurch kann ein bösartiger Peer keine gültige Signatur ohne den entsprechenden Schlüssel generieren. Gleiches gilt für die Verschlüsselung und Entschlüsselung von Daten. Zudem ist es einem bösartigen Peer zwar möglich, den Netzwerkverkehr abzuhören, jedoch kann er die Kommunikation zwischen zwei Peers nicht blockieren.

Die von PACS verwendeten Techniken ermöglichen, die verlässliche Speicherung zu garantieren, solange 70% der Peers im Netzwerk gutartig sind. Gründe für diese Schranke sind zunächst einmal technischer Natur. So funktioniert der Fehlertest, den die von PACS verwendete DHT einsetzt, nur bis zu dieser Grenze zuverlässig (siehe Kapitel 3.5.7.3). Auch bei der clientseitigen Durchsetzung muss sichergestellt sein, dass die Mehrzahl der Peers gutartig ist. Diese Schranke ist allerdings für die aufgezeigten Anforderungen mehr als ausreichend. Alle Peers sind durch eine zentrale Zertifizierungsinstanz registriert und können, falls sie eines bösartigen Verhaltens überführt werden, vom Netzwerk ausgeschlossen werden. Zudem erscheint es sehr fraglich, ob sich ein Netzwerk mit 30% bösartigen Peers zum gegenseitigen Austausch vertraulicher Daten eignet. In diesen Fällen sollte wohl besser gänzlich auf einen Austausch von Daten verzichtet werden.

3.5.3 Angriffe gegen die Integrität des Privilegienspeichers

Aus den Ausführungen zu bösartigen Peers und den Anforderungen an einen verlässlichen Privilegienspeicher ergeben sich folgende Bedrohungen, denen der Privilegienspeicher bei seiner Integritätsprüfung begegnen muss.

Datenmanipulationen Ein bösartiger Peer wird versuchen, die Daten des Privilegienspeichers zu fälschen und zu seinen Gunsten zu manipulieren.

Replay Bei Replay-Angriffen versuchen bösartige Peers, veraltete Daten als aktuell darzustellen. Im Gegensatz zur Datenmanipulation verfälscht der Peer die Daten nicht, sondern liefert lediglich Daten, die zu diesem Zeitpunkt nicht mehr aktuell sind.

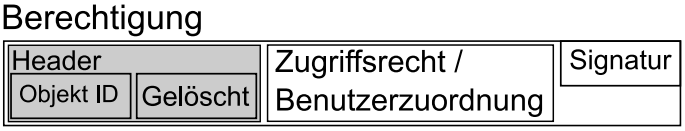


Abbildung 3.13: Aufbau Zugriffsrecht bzw. Benutzerzuordnung im Privilegienspeicher

Unterdrückung korrekter Daten Ein bösartiger Peer antwortet auf Anfragen nicht. Entsprechend werden die auf ihm gespeicherten Daten im Netzwerk nicht gefunden.

3.5.4 Schutz der Integrität des Privilegienspeichers

Der Privilegienspeicher dient der Speicherung globaler Zugriffsrechte und Benutzerzuordnungen. Diese werden im Folgenden zusammenfassend als Berechtigungen bezeichnet. Dazu bietet er über seine Schnittstelle drei Methoden nach außen an, die speziell der Verwaltung dieser Berechtigungen dienen. Diese sind $put(key, priv)$, $get(key)$ und $delete(key, priv)$. $put(key, priv)$ dient der Speicherung der Berechtigung $priv$ unter der ID key und $get(key)$ dient entsprechend dem Wiedererlangen der Berechtigung mit ID key. $delete(key, priv)$ bewirkt das Löschen der Berechtigung mit dem angegebenen Schlüssel. Wichtig ist festzustellen, dass der Schlüssel für eine Berechtigung hierbei nicht eindeutig sein muss. D.h. es kann mehrere Berechtigungen geben, die denselben Schlüssel besitzen. Ein Zugriffsrecht und eine Benutzerzuordnung sind hingegen immer eindeutig durch seinen Primärschlüssel bestimmt. Deshalb gibt die Funktion $get(key)$ potentiell eine Menge an Berechtigungen zurück. Da der Schlüssel nicht eindeutig ist, wird der Methode $delete(key, priv)$ auch noch die zu löschende Berechtigung als Parameter übergeben. Die Modifikation einer schon gespeicherten Berechtigung kann durch Einfügen einer modifizierten Berechtigung und Löschen der alten Berechtigung geschehen.

Die direkte Speicherung und Löschung von Privilegien und Benutzerzuordnungen aus dem Privilegienspeicher unterliegt ebenfalls der Privilegienverwaltung. Auch hierbei stellt sie die Korrektheit der Löschung bzw. Speicherung der Privilegien und Benutzerzuordnungen sicher. Daneben dient der Privilegienspeicher als Speicherort sonstiger sicherheitsrelevanter Daten. Die Verwaltung dieser Daten geschieht über die Methoden $put\_data(key, value)$, $get\_data(key)$ und $delete\_data(key)$. Sie dienen entsprechend der Speicherung, der Abfrage und dem Löschen von Daten. Wichtig ist hierbei, dass im Gegensatz zu Privilegien der Schlüssel eindeutig ist. Entsprechend gibt $get\_data(key)$ immer nur genau ein Datenobjekt zurück. Deshalb werden für die Verwaltung normaler Datenobjekte separate Methoden in der Schnittstelle nach außen zur Verfügung gestellt. Beim Einfügen und Löschen dieser Daten übernimmt die Privilegienverwaltung die Prüfung der Integrität dieser Daten. Diese Integritätsprüfungen sind vom Typ der gespeicherten Daten abhängig und werden bei der Erläuterung der Datentypen erörtert.

Die Integrität der im Privilegienspeicher gespeicherten Privilegien basiert einerseits auf dem Schutz der einzelnen Privilegien, andererseits auf dem Erkennen fehlerhafter oder ungültiger Privilegien.

3.5.4.1 Schutz der individuellen Berechtigung

Der Aufbau einer Berechtigung ist in Abbildung 3.13 schematisch dargestellt. Wie ersichtlich, besitzt jede Berechtigung einen Header, der neben einem eindeutigen Objektbezeichner (Objekt ID) ein Gelöscht-Flag enthält. Dieses Gelöscht-Flag zeigt an, ob die Berechtigung gelöscht

wurde und dient, wie im nächsten Kapitel erläutert, der Erkennung von Replay-Angriffen. Die im Privilegienspeicher abgelegte Berechtigung enthält daneben das Privileg bzw. die Benutzerzuordnung mit den in Kapitel 3.3.2 beschriebenen Attributen. Da der Grantzähler des Grantors bei jedem neu vergebenen Privileg und bei jeder neu vergebenen Benutzerzuordnung erhöht wird, können neuere von älteren Berechtigungen unterschieden werden. Der Grantzähler dient so der Versionierung der Berechtigungen. Schließlich ist der Header und das Privileg bzw. die Benutzerzuordnung durch den Grantor signiert. Diese Signatur ist ebenfalls Bestandteil der im Privilegienspeicher abgelegten Berechtigung. Die Signatur schützt die Berechtigung hierbei vor Manipulationen. Nur falls eine gültige Signatur des Grantors für das Privileg vorhanden ist, wird dieses im Privilegienspeicher gespeichert und für Auswertungen beachtet. Eine Ausnahme bildet hier ein gelöschtes Privileg. Wie im Folgenden erläutert, kann dieses stellvertretend auch vom Peer des Grantors signiert werden.

3.5.4.2 Löschen einer globalen Berechtigung

Wie aus den Methoden des Privilegienspeichers ersichtlich, gibt es eine Anweisung zur Löschung einer Berechtigung. Die Verhinderung von Replay-Angriffen macht jedoch eine zweite Methode notwendig. Jede Berechtigung besitzt hierzu das Gelöscht-Flag, das die Löschung der Berechtigung anzeigt. Eine gelöschte Berechtigung kann so als gelöscht markiert werden, ohne dass sie aus dem Privilegienspeicher entfernt werden muss. In diesem Fall geschieht das Löschen einer Berechtigung durch die erneute Speicherung der Berechtigung mit aktiviertem Gelöscht-Flag. Da die Berechtigung bis auf das aktivierte Gelöscht-Flag mit der ursprünglichen Berechtigung identisch ist, wird die bestehende Berechtigung überschrieben. Bei Replay-Angriffen liefern bösartige Peers weiter die nicht als gelöscht markierte Berechtigung aus. Nur durch Speicherung der gelöschten Berechtigung kann ein solcher Angriff erkannt werden. Allerdings ist ein derartiges Löschen nur für den Grantor der Berechtigung möglich, denn neue Berechtigung sind durch den Grantor zu signieren. Will also ein Benutzer g eine Berechtigung ber entziehen, die er Benutzer b gewährt hat, wird beim kaskadierenden Widerrufen von ber nur die direkt durch den Grantor g gewährte Berechtigung ber als gelöscht markiert und entsprechend gespeichert. Die von ber abhängigen Berechtigungen ber' werden hingegen nicht als gelöscht markiert, sondern komplett aus dem Privilegienspeicher gelöscht. Dies geschieht durch die Methode $delete(key, ber')$. g ist zu dieser Löschung autorisiert da der Widerruf von ber, den Delegationspfad von ber' unterbrochen hat. Eine Speicherung der ber' Berechtigungen mit aktiviertem Gelöscht-Flag ist nicht nötig, da Replay-Angriffe mit diesen Berechtigungen durch den unterbrochenen Delegationspfad eindeutig erkannt werden können.

Eine weitere Besonderheit existiert beim kaskadierenden Löschen globaler Berechtigungen durch eine Änderung einer Export Policy. Auch hier fehlt eventuell der Schlüssel des Grantors, um die zu entziehende globale Berechtigung mit aktivierten Gelöscht-Flag zu signieren, denn die Änderung der Export Policy werden üblicherweise durch Systemadministratoren vorgenommen. Die Änderung der Export Policy erfordert eine Signatur derselben mit dem privaten Schlüssels des Peers. Der Systemadministrator, der diese Änderungen an der Export Policy vornimmt, muss deshalb im Besitz des privaten Schlüssel des Peers sein. Die zu löschenden Berechtigungen werden stellvertretend für den Grantor mit dem Schlüssel des Peers signiert. Da der Peer des Benutzers die übergeordnete Instanz des Benutzers darstellt, wird das Löschen des Privilegs von der übergeordneten Instanz, eben dem Peer, autorisiert. Auf eine Speicherung des gelöschten Privilegs kann in diesem Fall nicht verzichtet werden, da dieses Privileg direkt von einem lokalen Benutzer des Peers gewährt wurde. Dies kann ohne

die Speicherung als gelöschtes Privileg erfolgreich für Replay-Angriffe genutzt werden.

3.5.4.3 Erkennung ungültiger Berechtigungen

Manipulierte Berechtigungen werden durch die vorgestellten Schutzmechanismen für die individuelle Berechtigung erkannt und deshalb im Folgenden nicht weiter betrachtet. Die Herausforderung bei der Erkennung ungültiger Privilegien ist vielmehr die Feststellung von Replay-Angriffen und ungültigen Privilegien. „Gültig“ bezieht sich hierbei auf die Korrektheit der Berechtigung gemäß der Überprüfung des globalen Privilegs bzw. der globalen Benutzerzuordnung. Sie sind gültig, falls für sie eine Delegationskette existiert. Dies wird durch die bereits definierten Funktionen *check_zugr* bzw. *check_bz* überprüft.

Wenn sich nur gutartige Peers im Netzwerk befinden, sind aufgrund der Annahme, dass jeder Peer eine Überprüfung der von ihm zu speichernden Berechtigungen vornimmt und deshalb nur gültige Berechtigungen speichert, keine ungültigen Berechtigungen im Privilegienspeicher vorhanden. Diese Annahme ist allerdings nicht realistisch und so muss mit ungültigen Berechtigungen im Privilegienspeicher gerechnet werden. Allerdings kommen ungültige Berechtigungen immer von bösartigen Peers, da die von gutartigen Peers durchgeführte Überprüfung ungültige Berechtigungen erkennt und entsprechend nicht speichert.

Eine Überprüfung des Delegationspfades kann sehr aufwändig sein, da sie zusätzliche $get(key)$ Aufrufe an den Privilegienspeicher beinhaltet. Wie viele dieser Aufrufe hierzu nötig sind und wie aufwändig diese sind, hängt von der Organisation und Art des Privilegienspeichers ab. Als alternatives Verfahren kann die Erkennung falscher oder veralteter Privilegien basierend auf der Replikation von Privilegien vorgenommen werden.

3.5.4.4 Überprüfung von Berechtigungen durch Stichprobentest

Die Replikation von Berechtigungen ist Teil der Techniken zur verlässlichen Speicherung von Privilegien. Berechtigungen werden statt auf einem Peer auf *num_rep* Peers gespeichert. Die Anzahl der Replikate ist bei PACS auf 32 festgelegt (siehe Kapitel 3.5.7.3). Allerdings muss die Gruppe der Peers, die die Replikate speichern, durch eine vertrauensvolle Autorität festgelegt worden sein. Dies garantiert die DHT, da die Gruppe der Peers, die Replikate eines bestimmten Datenobjekts bzw. Berechtigungen speichern, durch das zugrundeliegende Protokoll exakt definiert ist. Zudem haben die Peers selbst keinen Einfluss auf die Kriterien, die diese Festlegung bestimmen. Konkret haben die Peers in der DHT keinen Einfluss darauf, an welcher Position sie im Netzwerk eingefügt werden. Dies geschieht in PACS, wie später in Kapitel 3.5.7.1 beschrieben, durch die im Zertifikat festgelegte ChordID. Die Replikate werden von einem Peer immer auf den (in der Anordnung im Netzwerk gesehen) *num_rep* nächsten Peers gespeichert.

Für unstrukturierte Netzwerke ist der Dateneigentümer die Autorität, die eine Gruppe von Peers auswählt, die dann für die Speicherung der Replikate seiner Datenobjekte und der auf diesen gewährten Berechtigungen zuständig sind. Der Dateneigentümer stellt durch geeignete Auswahl sicher, dass der Anteil an bösartigen Peers in dieser Gruppe den Anteil der bösartigen Peers im Netzwerk nicht übersteigt. Das Vertrauen ist allerdings immer zweckgebunden. So ist die Gruppe, die vom Dateneigentümer definiert wurde, nur vertrauenswürdig bezüglich der Speicherung von Datenobjekten und Berechtigungen an Datenobjekten des Dateneigentümers. So wird verhindert, dass Replikationsgruppen, die von bösartigen Peers definiert wurden und so ausschließlich bösartige Peers enthalten, für die Speicherung fremder Datenobjekte

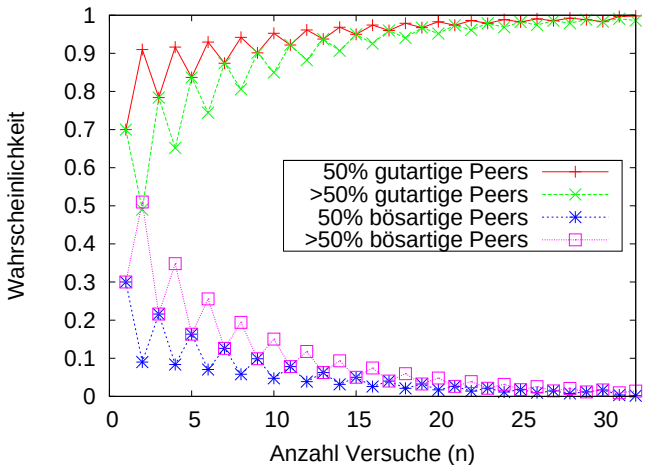


Abbildung 3.14: Dominanz bei unterschiedlicher Stichprobengröße

zuständig sind. In diesem Fall schlägt die nun vorgestellte Prüfung fehl.

Ausgangslage der folgenden Betrachtung ist die Annahme, dass eine Anfrage an den Privilegienspeicher eine Multimenge von Berechtigungen *antworten* zurückliefert. Diese lässt sich in die Teilmultimenge der korrekten Antworten *korr_ant* und falschen Antworten $falsch\_ant$ unterteilen. Sei des Weiteren gegeben, dass maximal 30% der Peers im Netzwerk bösartig sind und die Replikate der Berechtigungen gleichförmig im Netzwerk verteilt sind. Bei einer Binomialverteilung mit 32 Proben ist es mit einer Wahrscheinlichkeit von 99,48% gegeben, dass die Proben von bösartigen Knoten nicht in der Mehrzahl sind. Mit einer Wahrscheinlichkeit von 98,62% ist die Mehrzahl der Proben von gutartigen Knoten.

Der gesamte Verlauf der Wahrscheinlichkeiten bei verschiedener Anzahl an Versuchen ist in Abbildung 3.14 dargestellt. Wie ersichtlich, weist die Wahrscheinlichkeitskurve ein charakteristisches Zickzackmuster auf. Dies liegt an den Übergängen zwischen gerader und ungerader Anzahl an Versuchen. Da immer nur eine ganzzahlige Anzahl an Knoten gut oder bösartig sein kann, steigt die Schwelle benötigter guter bzw. schlechter Proben immer sprunghaft, was sich in den Sprüngen der Wahrscheinlichkeitsverteilung wiederspiegelt. Bei der Anfrage aller Replikate beträgt die Wahrscheinlichkeit 99,48%, dass die Gruppe *korr_ant* mindestens gleich groß ist wie $falsch\_ant$.

Aus dieser Beobachtung wurde in PACS folgende Konfliktlösungsstrategie implementiert. Bei ungleichen Gruppengrößen wählt der Privilegienspeicher das größere Ergebnis als korrekt aus und gibt dieses an den Anfragenden zurück. Sind beide Gruppen gleich groß, gibt die Stichprobenprüfung hierfür kein eindeutiges Ergebnis. Deshalb muss die Gültigkeit der in der Menge enthaltenen Berechtigungen durch die Funktionen *check_zugr* bzw. *check_bz* überprüft werden.

Bei der Überprüfung der Antwortmenge *antwort* für das Privileg *priv* gilt es, zwei Fälle, bei der Aufteilung in gültige und ungültige Antworten zu unterscheiden.

Vorhandensein einer gelöschten Berechtigung In diesem Fall liegt ein Privileg *priv* mit aktiviertem Gelöscht-Flag in der Antwortmultimenge *antworten* vor. Alle anderen Antworten für das Privileg *priv* sind deshalb der Menge $falsche\_ant$ zuzuordnen. Das korrekte Ergebnis also eine leere Menge.

Abbildung 3.15: Die Struktur einer Export Policy im Privilegienspeicher

Erkennung ungültiger Berechtigungen In der Multimenge *antworten* ist kein als gelöscht markiertes Privileg enthalten. Die Erkennung eines gültigen Privilegs hängt demnach vom Verhalten der bösartigen Peers ab. Deshalb sind hier wiederum zwei Extremfälle zu unterscheiden.

Im ersten Fall antworten alle bösartigen Peers mit dem gleichen ungültigen Privileg *priv*. Da dieses Privileg ungültig ist, antworten alle gutartigen Peers mit einer leeren Menge. Gibt es mehr Antworten mit der leerer Menge als mit dem Privileg *priv*, ist nach der Konfliktlösungsstrategie die leere Menge das korrekte Ergebnis.

Im zweiten Fall antworten alle bösartigen Peers bei Anfragen für das Privileg *priv* gar nicht oder mit einer leeren Menge. Die gutartigen Peers antworten hingegen mit *priv*, sofern *priv* ein gültiges Privileg ist. Ist *priv* korrekt, gibt die Konfliktlösungsstrategie *priv* als Ergebnis zurück. Ist *priv* hingegen ungültig, ist die leere Menge das Ergebnis.

In allen Fällen, bei denen die bösartigen Peers nicht perfekt miteinander kooperieren und ihre Antworten entsprechend nicht abstimmen, ist das Ergebnis eine Mischung aus ungültigen Privilegien und leeren Mengen. Dies erleichtert der Konfliktlösungsstrategie die Auswahl des korrekten Ergebnisses, da die Teilmultimenge übereinstimmender Antworten von bösartigen Peers kleiner ist.

Die Erkennung ungültiger Privilegien ist somit mit sehr hoher Wahrscheinlichkeit durch Stichprobenprüfung möglich. Nur für rare Fälle, in denen die Korrektheit der Privilegien nicht eindeutig durch die Konfliktlösungsstrategie bestimmt werden kann, muss die Prüfung des Delegationspfades des Privilegs dessen Gültigkeit bestimmen. Es werden dann entsprechend nur die Privilegien zurückgegeben, die diese Prüfung bestehen.

3.5.4.5 Schutz der Export Policies

Die Export Policies der Peers werden, neben der lokalen Speicherung auf dem Peer selbst, auch noch im Privilegienspeicher abgelegt. So wird deren Verfügbarkeit sichergestellt, falls ein Peer sich zeitweise nicht im Netzwerk befindet.

Wie schon erläutert, kann die Export Policy aus mehreren XACML Policies bestehen. Diese sind alle vom erstellenden Peer signiert. Zur Speicherung werden die Policies deshalb zu einem Archiv *archiv* verpackt und dieses mit $put\_date(key, archiv)$ im Privilegienspeicher abgelegt. Die Struktur einer im Privilegienspeicher abgelegten Export Policy ist in Abbildung 3.15 zu sehen. Das Archiv erhält hierbei einen Header mit Versionsnummer und dem eindeutigen Bezeichner der Export Policy $ObjektID$. Der gesamte Inhalt des Archivs ist zusammen mit dem Header signiert. Die Signatur entspricht hierbei der Signatur des Peers. Der Schlüssel *key*, unter dem die Export Policy abgelegt wird, ist systemweit festgelegt und leitet sich aus dem eindeutigen Peernamen ab. Er entspricht dem im Header abgelegten eindeutigen Bezeicher $ObjektID$.

Gutartige Peers des Privilegienspeichers akzeptieren nur Export Policy Archive, die vom zugehörigen Peer signiert wurden und deren Versionsnummer höher ist, als die in der der-

zeitigen Export Policy. Durch die Signatur sind unberechtigte Manipulationen an der im Privilegienspeicher abgelegten Export Policy sofort erkennbar. Durch die Versionsnummer im Header des Archivs lassen sich zudem auch Replay-Angriffe zuverlässig erkennen.

3.5.5 Unstrukturiertes P2P-Netzwerk

In einem unstrukturierten Netzwerk findet das Auffinden der Datenobjekte wie bereits in 2.7.1 erläutert durch Absuchen des gesamten Netzwerks statt. Um das Auffinden von Daten auch bei einem sich ständig ändernden Netzwerk zu gewährleisten, werden diese repliziert. Je mehr Peers Daten speichern, desto höher ist die Verfügbarkeit der Daten. Durch Replikation steigt allerdings auch der Aufwand der Modifikation, Löschung und Erzeugung von Daten. Häufig wird die Datenreplikation in P2P-Netzwerken adaptiv betrieben, d.h. Daten werden nur repliziert, falls sie angefragt werden. Ein Überblick dazu findet sich in Cohen und Shenker [CS02].

Dies ist für die aufgezeigten Anforderungen von PACS allerdings ungenügend. Privilegien sollen auch dann verlässlich gespeichert sein und entsprechend wiedergefunden werden können, wenn sie noch nie angefragt wurden. Deshalb ist für PACS nur die sogenannte aktive Replikation von Privilegien gangbar. Diese verlangt, dass sich ein Peer aktiv um die Replikation von Datenobjekten kümmern muss.

Es bietet sich an, diese aktive Rolle dem Peer zuzuweisen, der das zu speichernde Datenobjekt erstellt hat, d.h. dem Dateneigentümer. Aber wer kümmert sich um die aktive Replikation des Datenobjekts, wenn der Dateneigentümer sich nicht mehr im Netzwerk befindet? Ein durch den Dateneigentümer gewählter Stellvertreter könnte diese Aufgabe übernehmen. Voraussetzung dafür ist ein sehr hohes Vertrauensverhältnis zwischen Dateneigentümer und Stellvertreter. Dies ist in P2P-Umgebungen eher unrealistisch. Es ist deshalb möglich, dass der Stellvertreter die aktive Replikation des Datenobjekts unterlässt und das Datenobjekt irgendwann aus dem Netzwerk verschwindet.

Um das nötige Vertrauen in den einzelnen Stellvertreter zu minimieren, muss die Aufgabe der aktiven Replikation auf alle Peers übertragen werden, die ein Replikat eines Datenobjekts speichern. Für die Umsetzung einer solchen gruppenbasierten aktiven Replikation gibt es zwei unterschiedliche Herangehensweisen. Die erste Herangehensweise besteht darin, jeden Peer individuell die Verfügbarkeit der Replikate überprüfen zu lassen. Ein möglicher Ansatz, der sich an der Arbeit von Cuenca-Acuana u.a. [CAMN03] orientiert, sieht wie folgt aus:

Jeder Peer, der ein Replikat eines Datenobjekts speichert, überprüft periodisch die Verfügbarkeit des Replikats im Netzwerk. Um dies zu überprüfen startet der Peer p, der ein Replikat des Datenobjekts o besitzt, eine Abfrage nach diesem Datenobjekt im Netzwerk. Die Reichweite der Abfrage kann hierbei eingeschränkt werden. Bei Flooding kann z.B. die Anzahl der Weiterleitungen oder die „Time to Live“ der Anfrage entsprechend begrenzt sein. Anhand der erhaltenen Antworten auf eine solche Anfrage kann Peer p abschätzen, wie viele Replikate im gesamten Netzwerk zur Verfügung stehen. Hierzu setzt er die Reichweite seiner Anfrage mit den erhaltenen Antworten ins Verhältnis. Zusätzlich ist noch der von ihm geschätzte Anteil an bösartigen Peers mit einzubeziehen. Sind nach dieser Abschätzung zu wenige Replikate des Datenobjekts o im Netzwerk, wird p andere ihm bekannte Peers auffordern, ein Replikat von o zu speichern. Stellt Peer p bei seiner Anfrage fest, dass sein Replikat veraltet ist, wird er das von ihm gespeicherte alte Replikat o löschen und die Replikation dieses Objekts nicht weiter betreiben.

Die Hauptherausforderung bei einem solchen Vorgehen ist es, eine geeignete Funktion für

die Abschätzung der vorhandenen Replikate im Netzwerk zu finden. Bei dieser Lösung ist ein permanenter administrativer Aufwand durch das periodische aktive Abfragen der Datenobjekte vorhanden.

Bei der zweiten Herangehensweise wird eine Gruppe von Peers definiert, die Replikate eines Datenobjekts speichert. Die Gruppe der Peers verwaltet sich selbst und sorgt so autonom für die aktive Replikation der von der Gruppe gespeicherten Datenobjekte. Das periodische Abfragen der Datenobjekte entfällt hierdurch. Allerdings besteht der Aufwand der Gruppenverwaltung.

In PACS wurde die zweite Herangehensweise realisiert. Grund hierfür ist, dass auch für die clientseitige Durchsetzung der Privilegien eine definierte Gruppe von Peers benötigt wird, dort Stellvertreter genannt, (Details siehe Kapitel 3.8). Da diese Gruppe der Stellvertreter auch für die Speicherung von Privilegien eingesetzt werden kann, ergänzen sich die beiden Ansätze sehr gut. Zudem wurde diese Gruppe von einer Autorität (dem Dateneigentümer) definiert und ist deshalb für die Speicherung der Replikate dieses Eigentümers verlässlich. Die in PACS gewählte Herangehensweise wird im Folgenden genauer vorgestellt.

3.5.6 Unstrukturiertes P2P-Netzwerk mit definierter Replikationsgruppe

Die aktive Replikation gespeicherter Datenobjekte in einer definierten Gruppe ist Inhalt dieses Kapitels. Nach der Darstellung des generellen Aufbaus des Netzwerks und der Kommunikationsarten innerhalb desselben, erfolgt die Auseinandersetzung mit den Details der Datenreplikation und der Gruppenorganisation.

3.5.6.1 Aufbau des Netzwerks

Jeder Knoten des unstrukturierten P2P-Netzwerks besitzt und verwaltet verschiedene Verbindungen zu anderen Peers des Netzwerks. Die Verbindungen zu anderen Peers werden in einer in PACS als Fingertabelle[1] bezeichneten Datenstruktur gespeichert. Da es sich um ein unstrukturiertes Netzwerk handelt, werden die Knoten, zu denen Verbindungen gespeichert werden, zufällig ausgewählt.

Für Suchanfragen in dem so organisierten Netzwerk kommen die gängigen Abfragetechniken für unstrukturierte P2P-Netzwerke zum Einsatz. PACS arbeitet mit der Flooding-Technik. Ausschlaggebend hierfür ist ihre Robustheit gegenüber einem hohen Anteil von bösartigen Peers im Netzwerk. Eine Anfrage wird in PACS durch ihre maximale Anzahl an Weiterleitungen beschränkt, um die Anzahl der nötigen Anfragen nicht zu groß werden zu lassen. Zudem erhält jede Anfrage eine History, um redundante Weiterleitungen zu vermeiden. Das Routing einer Nachricht durch einen Peer erfolgt gemäß dem in Abbildung 3.16 dargestellten Pseudocode.

3.5.6.2 Verwaltung der Fingertabelle

Die Einträge in den Fingertabellen der Knoten müssen regelmäßig auf ihre Gültigkeit überprüft werden. Hierzu versenden die Knoten sogenannte Alive-Anfragen. Antwortet ein Knoten auf eine solche Nachricht nicht, betrachtet der anfragende Knoten ihn als nicht länger

[1] Abgeleitet von der in Chord als „finger table“ bezeichneten Datenstruktur, die die Verbindungen zu anderen Knoten beinhaltet

```
private void broadcastMessage(Message msg){
  if(!msg.reachedMaxHops()&&!msg.isTimedOut()){
    Message send = new Message(msg); //Klonen der Nachricht
    send.addToHistory(finger); //Hinzufügen der Knoten zur History
    send.incHopCount(); //Hop Zähler erhöhen
    for (NodeHandle currHandle: finger){
      if (!msg.isInHistory(currHandle)){
        sendMessage(currHandle,send);
      }
    }
  }
}
```

Abbildung 3.16: Routing durch Flooding im unstrukturierten P2P-Netzwerk

erreichbar. Der Eintrag des nicht antwortenden Peers wird deshalb aus der Fingertabelle des anfragenden Knoten gelöscht.

Einträge werden ebenfalls aus der Fingertabelle gelöscht, wenn beim Senden einer Nachricht an einen Peer ein Fehler auftritt. Ist der Empfänger nicht mehr im Netzwerk erreichbar, erhält der sendende Knoten eine Mitteilung über die Nichtzustellbarkeit der Nachricht. Eine solche Rückmeldung führt ebenfalls zum Löschen des betroffenen Peers aus der Fingertabelle.

Da sich die Anzahl der Fingereinträge durch die Eliminierung nicht mehr erreichbarer Knoten reduziert, muss die Fingertabelle regelmäßig mit neuen Einträgen aufgefüllt werden. Hierzu wird von den noch bestehenden Fingereinträgen ein Knoten zufällig ausgewählt, der dann einen zufälligen Eintrag seiner Fingertabelle zurückliefert.

3.5.6.3 Beitritt von Knoten zum Netzwerk

Will ein neuer Knoten dem Netzwerk beitreten, muss er zunächst mindestens einen Peer des Netzwerks kennen, der ihm den ersten Zugang zum P2P-Netz ermöglicht. Peers, die diesen ersten Zugang gewähren, heißen Bootstrap-Knoten. An diese Bootstrap-Knoten sendet der beitretende Peer FingerEntry-Anfragen zum Auffinden weiterer Peers. Jeder Peer, der eine FingerEntry-Nachricht empfängt, antwortet dem beitretenden Peer und leitet die Nachricht gleichzeitig gemäß dem schon beschriebenen Flooding Algorithmus (siehe Abbildung 3.16) weiter. Dieses Verhalten trifft allerdings nur für gutartige Knoten zu. Bösartige Knoten werden, um andere gutartige Knoten unter Kontrolle zu bringen, zwar antworten, die Anfrage aber nur an andere bösartige Knoten weiterleiten. Die Antworten werden beim beitretenden Peer gesammelt. Aus den gesammelten Antworten wählt der beitretende Peer seine Fingereinträge aus und ist damit dem Netzwerk beigetreten.

Die Auswahl der Bootstrap-Knoten ist kritisch. Werden lediglich bösartige Knoten als Bootstrap-Knoten ausgewählt, kann der beitretende Peer keine gutartigen Peers kontaktieren. Somit unterliegt er der Kontrolle der bösartigen Knoten.

Für neu beitretende Knoten muss es also eine Möglichkeit geben, andere Peers kennenzulernen, um die entsprechenden Bootstrap-Knoten gezielt auszuwählen. Diese Aufgabe übernimmt in PACS die zentrale Zertifizierungsinstanz. Diese muss vom beitretenden Peer ohnehin zur Ausstellung der Zertifikate für den Peer und seine Benutzer kontaktiert werden. Die Zertifizierungsinstanz bietet entsprechend eine Liste mit Peers im Netzwerk an, die dem Peer als

Bootstrap-Knoten dienen können.

3.5.6.4 Replikationsgruppen

Voraussetzung für eine Replikation mit einer definierten Replikationsgruppe ist, dass jeder Knoten eine solche definierte Gruppe besitzt. Sobald der Knoten dem Netzwerk beigetreten ist, kann er sich eine Replikationsgruppe erstellen und auch selbst Mitglied in einer Replikationsgruppe werden. Die Verwaltungsinformationen zu einer Replikationsgruppe werden in einem Delegation Group Information Object (DGIO) abgelegt. Hier ist insbesondere festgelegt, welche Peers zu der Gruppe gehören und wer Eigentümer dieser Gruppe ist. Dieses DGIO muss für jeden Peer des Netzwerks auffindbar sein. Entsprechend wird es durch die Replikationsgruppe selbst repliziert und gespeichert. Der Schlüssel des Datenobjektes leitet sich direkt von dem eindeutigen Peernamen ab. So kann das DGIO eines Peers durch jeden Peer im P2P-Netzwerk gesucht werden. Ein Peer kann also die Replikationsgruppen anderer Peers finden und diese ebenfalls unter dem von seinem Peernamen abgeleiteten Schlüssel im Netzwerk registrieren[2] oder seine eigene Replikationsgruppe erstellen.

Den Peers einer Replikationsgruppe werden alle Gruppenmitglieder vorgestellt, sodass sich alle Peers gegenseitig kennen. Eine Replikationsgruppe verwaltet sich selbst. Der Eigentümer der Gruppe muss sich deshalb nicht ständig im Netzwerk befinden. Die genauen Protokolle zur Verwaltung einer solchen Gruppe, insbesondere wie und unter welchen Bedingungen die Gruppe verändert wird, werden ausführlich in Kapitel 3.8 beschrieben. Für das allgemeine Verständnis genügt es, davon auszugehen, dass eine Änderung an einer Gruppe immer nur in Abstimmung mit allen Gruppenmitgliedern geschehen kann. Es ist zudem Aufgabe aller Gruppenmitglieder, gegenseitig die Verfügbarkeit der anderen Gruppenmitglieder periodisch zu prüfen. Im Gegensatz zur aktiven Replikation ohne definierte Replikationsgruppe wird hier nicht das Vorhandensein des Datenobjekts überprüft, sondern immer nur die Erreichbarkeit des Peers. Deshalb kann diese Überprüfung durch direkte Kommunikation zwischen den Peers stattfinden. Eine Suche im Netzwerk ist hierfür nicht nötig. Wie im Falle der Fingertabelle senden sich die Peers für diese Überprüfung Alive-Nachrichten. Falls ein Peer auf diese Nachrichten nicht antwortet, gilt er als nicht mehr erreichbar. Der Peer, der dieses feststellt (im Folgenden Manager Peer genannt), führt dann den Algorithmus zur Gruppenänderung durch, der aus den folgenden Schritten besteht:

1. Die verbleibenden Peers der Replikationsgruppe werden vom Manager Peer über die Abwesenheit des Gruppenmitglieds unterrichtet. Dies geschieht durch Übermittlung einer Signature-Anfrage. Bestandteil dieser Nachricht ist auch ein Vorschlag für ein Ersatzgruppenmitglied.

2. Die so benachrichtigten anderen Gruppenmitglieder überprüfen die Abwesenheit des angegebenen Peers. Zugleich überprüfen sie die Erreichbarkeit des vorgeschlagenen Ersatzpeers. Sind beide Überprüfungen erfolgreich (d.h. der angegebene Peer antwortet auf die Alive-Anfrage nicht und der Ersatzpeer hat die Alive-Anfrage beantwortet) und erscheint den übrigen Peers der Ersatzpeer als neues Gruppenmitglied geeignet, werden diese die Änderungen an der Gruppe bestätigen. Dies geschieht durch die Beantwortung der Signature-Nachricht des Manager Peers.

[2]Das DGIO besitzt in diesem Fall mehrere Schlüssel unter denen es gefunden werden kann.

3. Nachdem der Manager Peer die Signature-Antworten der anderen Gruppenmitglieder erhalten hat, ändert er die Gruppenverwaltungsinformationen im DGIO entsprechend und übermittelt das neue DGIO an die Gruppenmitglieder.

4. Die alten Gruppenmitglieder senden nun sogenannte Share-Nachrichten an den neuen Peer der Gruppe. Diese für die Gruppenverwaltung wichtigen Informationen benötigt der neue Peer, um ein vollständiges Mitglied der Replikationsgruppe zu werden.

5. Der Manager Peer sendet die Replikate, die diese Gruppe speichert, an den Ersatzpeer.

Durch dieses Vorgehen ist gewährleistet, dass immer die durch die Gruppengröße definierte Anzahl an Replikaten der Datenobjekte im Netzwerk vorhanden ist.

Bösartige Peers stellen für dieses Verfahren keine Gefahr dar, da die Abstimmung über den Ausschluss eines Peers aus der Gruppe und die Wahl eines Ersatzpeers durch alle Gruppenmitglieder vorgenommen wird (siehe Kapitel 3.8). Der Algorithmus zur Gruppenänderung kann mehrere Änderungen an der Gruppe gleichzeitig vornehmen. Entsprechend sind die Vorgänge, die das verschwundene oder neue Gruppenmitglied betreffen, für alle verschwundenen bzw. neuen Gruppenmitglieder durchzuführen.

3.5.6.5 Datenoperationen

Die nach außen vom Privilegienspeicher angebotenen Methoden $put(key, priv)$, $get(key)$ und $delete(key, priv)$ für die Privilegienverwaltung und die Methoden für die Speicherung sonstiger Daten $put\_data(key, value)$, $get\_data(key)$ sowie $delete\_data(key)$ sind entsprechend auch auf das unstrukturierte Netzwerk abgebildet. Ein $put(key, value)$ und $put\_data(key, value)$ speichert das Datenobjekt bzw. Berechtigung $value$ unter der ID key im lokalen Privilegienspeicher des Peers. Für die Replikation der Datenobjekte in der Replikationsgruppe sorgt entweder der anfragende Peer oder der Eigentümer der Replikationsgruppe. Gutartige Peers, die eine solche Nachricht empfangen, werden diese Datenobjekte ebenfalls speichern.

Die Operation $delete(key, value)$ löscht die Berechtigung $value$ mit der ID key aus dem Privilegienspeicher (entsprechendes gilt für die Operation $delete\_data(key)$). Die Löschung der Replikate obliegt wiederum dem Dateneigentümer oder dem anfragenden Peer. Hierbei werden Delete-Nachrichten an die Peers der Replikationsgruppe versendet, um auch dort eine Löschung des Datenobjekts sicherzustellen. Bösartige Peers werden dieser Aufforderung nicht nachkommen.

Die Methoden $get(key)$ und $get\_data(key)$ initiiert eine Suche nach key im Netzwerk. Hierbei kommt die schon beschriebene Flooding-Technik zum Einsatz. Jeder Peer, der eine solche Lookup-Nachricht erhält, führt neben der Weiterleitung der Nachricht eine lokale Anfrage durch. Ist ein Datenobjekt mit entsprechendem Schlüssel im lokalen Privilegienspeicher, wird dieses zurückgegeben. Bösartige Peers liefern bei Lookup-Anfragen entweder ein gefälschtes oder veraltetes Privileg zurück oder antworten nicht. Zudem werden sie die Nachricht nur an andere kooperierende bösartige Peers weiterleiten.

Durch die hier vorgestellte und für PACS entwickelte aktive Replikation mit definierter Replikationsgruppe kann das Auffinden eines gespeicherten Datenobjekts sichergestellt werden, solange die Anzahl der Routingschritte groß genug gewählt wurde. Insbesondere wird sichergestellt, dass ein Datenobjekt nicht verloren geht, auch dann, wenn der Dateneigentümer sich nicht im Netzwerk befindet. Somit lässt sich das Unterdrücken korrekter Daten durch bösartige Peers sicher verhindern.

3.5.7 Verteilte Hashtabelle (DHT)

Bei der Verwendung einer DHT zur Speicherung der Daten sind die Modifikation gegenüber einer normalen DHT wesentlich umfangreicher, als dies in einem unstrukturierten P2P-Netzwerk nötig ist.

Grundvoraussetzung ist hier die verlässliche Zuweisung der Peernamen (PeerIDs), um Sybil-Angriffe abzuwehren und die Replikation der gespeicherten Daten sicherzustellen. Da eine DHT auf einer Routinginfrastruktur aufbaut, muss auch diese vor einer überproportionalen Beeinflussung durch bösartige Peers geschützt werden. Dies kann in die Teilaufgaben sichere Verwaltung der Routinginformationen und verlässliches Weiterleiten der Nachrichten aufgeteilt werden. Bei der Lösung dieser Teilaufgaben orientiert sich der hier vorgestellte Ansatz an den Arbeiten von Castro u.a. [CDG+02] sowie Wallach [Wal02]. Diese Arbeiten beschreiben Lösungen für beide Teilaufgaben. Auch wenn die dort vorgestellten Konzepte größtenteils allgemein formuliert sind, wird deren konkrete Realisierung nur für Pastry gezeigt. PACS verwendet hingegen Chord als zugrundeliegende Routing Infrastruktur der DHT. Dafür spricht folgendes:

- Im Chord Protokoll ist strikt festgelegt, zu welchen Knoten Verbindungen in der Fingertabelle gespeichert werden. Dies ist nötig, um einer Manipulation durch bösartige Peers vorzubeugen. Bei anderen Protokollen muss dies erst mühsam etabliert werden.

- Chord macht keinerlei Annahmen bezüglich der räumlichen Verteilung der Knoten und deren IP-Adressen. Entsprechend müssen die von der zentralen Zertifizierungsinstanz vergebenen ChordIDs nicht an feste IP-Adressen gebunden werden.

Das Chord Protokoll bietet deshalb wenig Angriffsfläche für Manipulation. Die notwendigen Änderungen und Erweiterungen am Protokoll lassen sich in der Folge auf ein realistisches Maß reduzieren. Das so erweiterte Chord Protokoll wird als *verlässliches Chord* und entsprechend die DHT als *verlässliche DHT* bezeichnet.

3.5.7.1 Festlegung der ChordID

Wie bereits ausgeführt, wird die ChordID von der Zertifizierungsinstanz nach dem Zufallsprinzip vergeben (z.B. durch konsistentes Hashing des gesamten Zertifikats) und im Zertifikat des Peers gespeichert. Die ChordID des Peers bleibt dadurch dauerhaft mit dem Namen des Peers verbunden. Auf die Speicherung der IP-Adresse des Peers im Zertifikat kann verzichtet werden, da Chord die räumliche Verteilung der Peers anhand der IP-Adresse nicht berücksichtigt.

3.5.7.2 Verwaltung der Routinginformationen

Die sichere Verwaltung der Fingertabelle ist Basis für die Realisierung der verlässlichen Speicherung. Hierbei muss sichergestellt werden, dass der Anteil an Einträgen in der Fingertabelle zu bösartigen Peers (auch schlechte Fingereinträge genannt) nicht den Anteil der bösartigen Peers im gesamten Netzwerk übersteigt. Sollte es bösartigen Peers gelingen, einen größeren Anteil der Fingertabelle mit kooperierenden bösartigen Peers zu besetzen, sinkt die Wahrscheinlichkeit, dass Nachrichten von diesem Peer an andere gutartige Peers weitergeleitet werden. Somit leidet die Erreichbarkeit der Knoten im Netzwerk und damit die Verfügbarkeit

Notation	Definition
$n.finger[k]$	*k-te* Fingereintrag der Fingertabelle von Knoten n, wobei $1 \leq k \leq 32$
$n.finger[k].id$	$n + 2^{(i-1)} \pmod{2^{32}}$
$n.finger[k].knoten$	$successor(n.finger[k].id)$
$n.successor$	$n.finger[1].knoten$ Der Nachfolgende Knoten im ID Raum (Nächster Knoten im Uhrzeigersinn von ID n aus)
$n.predecessor$	Der Vorgängerknoten im ID Raum (Nächster Knoten entgegen dem Uhrzeigersinn von ID n aus)
$successor(id)$	Gibt den erste Knoten im Uhrzeigersinn von ChordID id aus zurück (Ein Knoten mit ChordID id ist hierbei auch $successor(id)$)
$bad\_successor(id)$	Gibt den nächsten *bösartigen* Knoten im Uhrzeigersinn von ChordID id aus zurück
$n.succ$	Die Successor-Liste von Knoten n
$n.rep$	Die Replikationsgruppe von Knoten n ($n.rep \subseteq n.succ$)
$n.pred$	Die Predecessor-Liste von Knoten n

Tabelle 3.4: Notationen Chord

der Daten. Anders ausgedrückt muss davon ausgegangen werden, dass bösartige Peers versuchen, den gutartigen Peers andere kooperierende bösartige Peers als Kandidaten für Einträge der Fingertabelle anzubieten. Durch jeden zusätzlichen schlechten Eintrag in der Fingertabelle steigt die Wahrscheinlichkeit, dass Anfragen an einen bösartigen Peer weitergeleitet und von ihm beantwortet werden.

Wenn im Folgenden von Einträgen oder Verweisen auf andere Knoten die Rede ist, bestehen diese immer aus der ChordID des Peers, auf den verwiesen wird, und der IP-Adresse, über die mit dem Peer kommuniziert werden kann.

Die erste Maßnahme zur Absicherung der Fingertabelle besteht darin, Bedingungen für die Fingertabelleneinträge festzulegen.

Bedingungen für Einträge der Fingertabelle In Chord haben Fingereinträge vom Protokoll festgelegte ChordIDs. In PACS ist der Raum der ChordID von 0 bis $2^{32} - 1$ definiert. Die Fingertabelle enthält deshalb 32 Einträge. Bei der Fingertabelle des Peers mit der ChordID n beinhaltet der *i-te* Fingereintrag den Successor für die ChordID $n + 2^{(i-1)} \pmod{2^{32}}$, wobei $1 \leq i \leq 32$. Der Successor von ID id kann hierbei mit der Funktion $successor(id)$ bestimmt werden. Chord ordnet die Peers entsprechend ihrer IDs ringförmig an und Nachrichten werden nur im Uhrzeigersinn weitergeleitet. Eine graphische Darstellung dieser Anordnung und der Fingertabellen ist in Abbildung 3.17 dargestellt. Zur übersichtlicheren Darstellung dienen die Notationen der Tabelle 3.4. Trotz der genau geregelten Vorgaben für die Einträge der Fingertabellen versuchen bösartige Peers, Fingertabelleneinträge mit bösartigen kooperierenden Knoten zu besetzen. Um dies zu erreichen, müssen sie dafür sorgen, dass ein Peer n nichts von vorhandenen guten Knoten zwischen $n.finger[k].id$ und $bad\_successor(n.finger[k].id)$ erfährt und so annimmt, der bösartige

Peer $bad\_successor(n.finger[k].id)$ ist gleich $successor(n.finger[k].id)$. Solche manipulierten Fingereinträge verweisen dauerhaft nicht auf den gemäß der Anordnung der ChordIDs im Chord Ring vorgesehenen Peer. Da $n.finger[k] \neq successor(n.finger[k].id)$ wird hierdurch zunächst nur die Routinginfrastruktur gestört. Mit zunehmender Anzahl solcher manipulierten Fingereinträge wird letztlich aber der durch diese Routinginfrastruktur definierte Chord Ring zerstört. Ist dies geschehen, zerbricht das Chord Netzwerk in mehrere Subnetze und wird somit unbrauchbar.

Es ist die Aufgabe des verlässlichen Chord Protokolls, sicherzustellen, dass es den bösartigen Peers trotz dieser Versuche nicht gelingt, Fingereinträge dauerhaft zu manipulieren. Dies kann durch das verlässliche Weiterleiten der Nachrichten sichergestellt werden, das garantiert, dass der richtige Successor $successor(n.finger[k].id)$ gefunden wird. Die Beschreibung dieser verlässlichen Weiterleitung folgt nach den anschließenden Ausführungen zu den Successor- und Predecessor-Listen.

Liste mit Successor- und Predecessor-Knoten Neben der Fingertabelle besitzen Peers in PACS noch zwei zusätzliche Datenstrukturen, die Informationen zu den anderen Teilnehmern des Netzes verwalten. Zum einen die Successor-Liste *succ*. Sie enthält die *sa* nächsten Successor-Peers des Knoten. Diese Successor-Liste dient der Replikation der gespeicherten Datenobjekte. Jeder Knoten ist der sogenannte Stammknoten für die Datenobjekte, für deren ChordID er Successor ist ($stamm(id) = successor(id)$, wobei id die ChordID des Datenobjekts ist). Der Stammknoten ist für die Replikation der Datenobjekte im Netzwerk verantwortlich. Der Stammknoten wird deshalb die Datenobjekte, für die er Stammknoten ist, an seine ar nächsten Successors zur Speicherung weiterleiten. Diese ar Successors bilden die Replikationsgruppe *rep* des Knotens. Die Replikationsgruppe ist immer eine Teilmenge der Successor-Liste ($rep \subseteq succ$ und $ar \leq sa$).

Zum anderen besitzt jeder Peer eine Predecessor-Liste *pred*. Diese enthält die ar nächsten unmittelbaren Vorgänger des Knotens. Sie wird für die Verwaltung der Replikationsgruppe benötigt, weshalb sie auch dieselbe Größe besitzt.

Sowohl die Successor-Liste als auch die Predecessor-Liste sind gegenüber bösartigen Angriffen nicht gefährdet, denn beide leiten sich aus den Einträgen der eigenen Fingertabelle und der Fingertabellen der Successor- bzw. Predecessor-Peers des Knotens ab. Entsprechend können Manipulationen an der Successor-Liste erkannt werden, indem die dort enthaltene Information mit dem Inhalt der eigenen Fingertabelle und Rückfragen bei benachbarten Peers verglichen wird. Zudem spiegelt die Liste die Anordnung der Knoten im Netzwerk wieder. Diese ist hierbei durch die ChordID der Peers vorgegeben. Eine Manipulation dieser Anordnung muss mit einer permanenten Verdrängung aktiver benachbarter Knoten einhergehen, was lediglich bei einer vollständigen Kontrolle dieser Peers möglich ist. Die Übernahme einer solchen vollständigen Kontrolle wird aber durch die sichere Verwaltung der Fingertabelle und die verlässliche Weiterleitung der Nachrichten verhindert.

Für bösartige kooperierende Peers ist die Manipulation der Fingertabelleneinträge eines gutartigen Knotens wesentlich erfolgversprechender. Erst nachdem die Manipulation der Fingertabelle entsprechend fortgeschritten ist, ist es sinnvoll mit der Manipulation der Successor- und Predecessor-Liste fortzufahren.

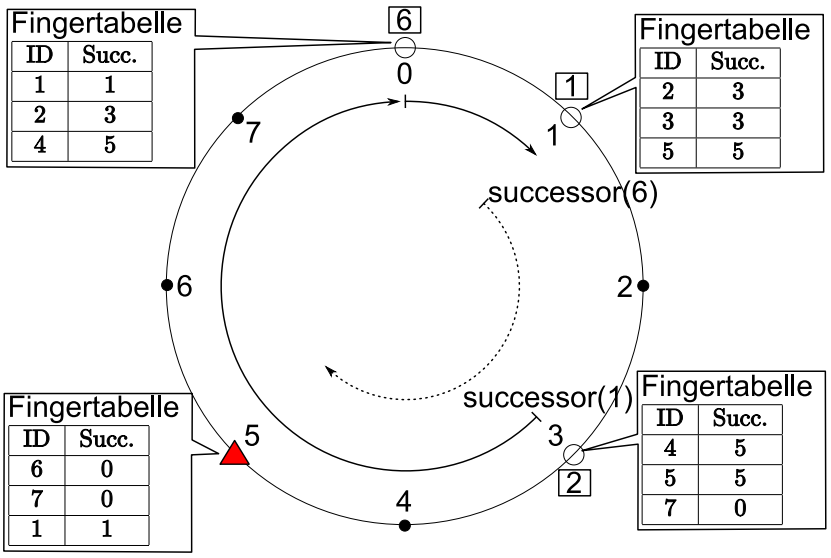


Abbildung 3.17: Chord Routing mit bösartigen Knoten

3.5.7.3 Verlässliches Weiterleiten von Nachrichten

Ein verlässliches Weiterleiten von Nachrichten stellt sicher, dass eine Nachricht zur Suche der ChordID *id*, die über das Chord Netzwerk geroutet wird, auch ihr Ziel *successor*(*id*) erreicht. Sind alle Peers im Netzwerk gutartig, ist dies durch das Chord Protokoll garantiert. Bösartige Peers im Netzwerk können hingegen eine Nachricht entweder nicht weiterleiten, diese an einen falschen Peer (insbesondere andere kooperierende bösartige Peers) weiterleiten, oder vorgeben, selbst der Successor zu sein, und die Nachricht beantworten. Dies ist in Abbildung 3.17 dargestellt. Das Netzwerk wurde gegenüber dem Chord Netzwerk in Abbildung 2.6 um einen bösartigen Knoten mit ChordID fünf erweitert. Zur Darstellung zweier Chord Routing Vorgänge sind im Beispiel der ChordID Raum und die Anzahl der Fingereinträge beschränkt. Beim ersten Routing Vorgang sucht Knoten drei den *successor*(1). Gemäß seiner Fingertabelle leitet er deshalb die Nachricht an Knoten null weiter. Dieser leitet die Nachricht weiter an den Knoten eins den *successor*(1) und das Routing war erfolgreich. Die Anfrage von Knoten eins nach *successor*(6) scheitert hingegen, da Knoten eins die Nachricht gemäß seiner Fingertabelle zunächst an Knoten fünf weiterleitet. Bei Knoten fünf handelt es sich um einen bösartigen Knoten und so bricht das Routing hier ab.

Unter der Annahme, dass in den Fingertabellen der Peers anteilig nur so viele schlechte Einträge enthalten sind, wie bösartige Peers im Netzwerk sind, entspricht die Wahrscheinlichkeit, dass ein Peer eine Nachricht an einen bösartigen Peer weiterleitet, dem Anteil bösartiger Knoten im Netzwerk. Bei einer durchschnittlichen Anzahl an Routingschritten h liegt die Gesamtwahrscheinlichkeit, dass eine Nachricht nicht fehlgeleitet wurde und ihr Ziel somit erreicht, bei $(1-b)^{h-1}$, wobei b den Anteil bösartiger Peers darstellt. Allerdings kann der Zielknoten ebenfalls bösartig sein, sodass z.B. das Auffinden eines Datenobjekts nur mit einer Wahrscheinlichkeit von $(1-b)^h$ gelingt, falls keine Replikate vorhanden sind. Die Anzahl der durchschnittlich nötigen Routingschritte beträgt bei Chord $\log_2(N)/2$, wobei N die Anzahl der Knoten im Netzwerk darstellt. Bereits bei einer Netzwerkgröße von 1000 Knoten und einem Anteil von 30% bösartiger kooperierender Knoten ergibt sich somit eine Erfolgswahr-

```
public double avgIdDistance(NodeHandle node,Vector <NodeHandle>
                            nodeHandles){
  //Sortieren der Knoten im Uhrzeigersinn
  Vector <NodeHandle> sortedHandles =
    sortClockwise(node, new Vector<NodeHandle>(nodeHandles));
  double sumDistance = 0.0;
  NodeHandle one = null;
  NodeHandle two = null;
  int numValues = 0;
  for (int i = 0;i < sortedHandles.size();++i){
    //Initialisierung der zwei NodeHandles
    if (one==null){
      one = sortedHandles.get(i);
    }else if (two==null){
      two = sortedHandles.get(i);
    }else{
      one = two;
      two = sortedHandles.get(i);
    }
    //Berechnung der Distanz zwischen zwei Knoten
    if (one!= null && two!=null && !one.equals(two)){
      sumDistance += calcNodeDistance(one.getId(),two.getId());
      numValues++;
    }
  }
  return sumDistance/numValues;
}
```

Abbildung 3.18: Berechnung der Dichte einer Menge von Chord Knoten

scheinlichkeit von nur noch ca. 17%.

Castro u.a. [CDG+02] schlagen zur Lösung dieses Problems kontrolliertes Flooding durch Ausnutzung der bestehenden Routinginfrastruktur vor, in Kombination mit der Erkennung fehlgeleiteter Anfragen (und damit falscher Antworten) durch einen Fehlertest. Die Anpassung dieser Techniken an Chord und ihr Zusammenspiel werden im Folgenden genauer erläutert.

Fehlertest Der Fehlertest wurde von Castro u.a. [CDG+02] unverändert übernommen. Seine Idee basiert darauf, dass die Dichte der bösartigen Knoten im ChordID Raum immer geringer ist, als die Dichte aller Knoten. Die Dichte entspricht hierbei der durchschnittlichen Distanz zwischen den ChordIDs der Knoten. Die Berechnung der Dichte einer Menge von Knoten ist in Abbildung 3.18 gezeigt. Die Distanz zwischen zwei Knoten in Uhrzeigersinn wird hierbei durch die Funktion in Abbildung 3.19 berechnet.

Diese Beobachtung wird nun wie folgt ausgenutzt. Jeder Antwort auf eine Anfrage ist vom antwortenden Peer eine Gruppe von Peers anzufügen, die sogenannte Antwort Knoten Menge (AKM). Für diese Menge wird der Fehlertest durchgeführt. Zunächst wird die Knotenmenge überprüft. Alle Knoten der AKM müssen ein gültiges Zertifikat der zentralen Zertifizierungsinstanz besitzen. Zudem erfüllt die AKM die Anforderungen der jeweiligen Anfrage. D.h. es handelt sich um eine mögliche Replikationsgruppe bzw. Predecessor-Liste des antwortenden Peers. Die Dichte dieser Menge wird nun mit der Dichte der Knoten in der Successor-Liste

```
public static double calcNodeDistance(Id first, Id second){
  BigInteger firstValue = (BigInteger) first.getValue();
  BigInteger secondValue = (BigInteger) second.getValue();
  BigInteger diff = secondValue.subtract(firstValue);
  //diff ist negativ => Ende des Rings wurde überschritten
  if (diff.compareTo(BigInteger.ZERO)< 0){
    //Distanz von firstValue zu maxId
    diff = maxId.subtract(firstValue);
    //Füge die Distanz von Anfang des Rings bis secondValue hinzu
    diff = diff.add(secondValue);
  }
  return diff.doubleValue();
}
```

Abbildung 3.19: Berechnung der Distanz zwischen Knoten im Chord Ring

des Empfängerpeers n verglichen. Hierbei muss folgende Bedingung gelten:

$$avgIdDistance(AKM) < avgIdDistance(n.succ) \times \gamma$$

Der γ Wert bestimmt die Abwägung zwischen falsch positiven und falsch negativen Testergebnissen. Nach Berechnungen von Castro u.a. [CDG+02] ist $\gamma = 1{,}23$ ein guter Kompromiss, der mit sehr hoher Wahrscheinlichkeit keine falsch negativen Ergebnisse liefert.

Je nach Anfrage entspricht die AKM entweder der Replikationsgruppe oder der Predecessor-Liste des antwortenden Peers. Wichtig ist, dass die darin enthaltenen Peers die Anfrage ebenfalls beantworten können. Im Falle einer Datenanfrage muss der antwortende Peer n entsprechend seine Replikationsgruppe $n.rep$ übermitteln. Durch diese Maßnahme sind bösartige Peers, die eine falsche Antwort gegeben haben und den Anfragenden von der Richtigkeit der Antwort überzeugen wollen, gezwungen, eine Gruppe als AKM zu übermitteln, die ausschließlich aus bösartig kooperierenden Knoten besteht. Nur diese werden bei einem Stichprobentest die gleiche falsche Antwort zurückliefern.

Die Genauigkeit des Vergleichs erhöht sich, wenn die Mengen möglichst groß sind. Die Erhöhung der Anzahl der Replikate im Netzwerk ist mit erheblichem Aufwand verbunden. Deshalb wurde die Anzahl der Replikate von Castro u.a. [CDG+02] auf 32 beschränkt und entsprechend hier auch für PACS übernommen. Die Anzahl der Knoten in der Successor-Liste des Empfängers kann hingegen relativ leicht erhöht werden.

Die Annahme, dass bösartige Peers nur bösartige kooperative Knoten in ihrer AKM zurückliefern, setzt voraus, dass der anfragende Peer alle Knoten der AKM nochmals anfragt. Bei einer Anzahl von 32 Replikaten ist dies sehr aufwändig. Deshalb werden in PACS nicht alle Knoten kontrolliert, sondern nur eine Stichprobe genommen. Bösartige Knoten haben so die Möglichkeit, auch gutartige Knoten in die AKM zu mischen und darauf zu vertrauen, dass nicht alle Knoten überprüft werden. Die Wahrscheinlichkeiten, bei unterschiedlicher Anzahl an gutartigen Peers in der AKM einen guten Peer zu wählen und so die falsche Antwort zu erkennen, sind in Abbildung 3.20 dargestellt. Hierbei wurde die Stichprobengröße variiert. Da Stichprobentest und Fehlertest unabhängig voneinander sind können dieses miteinander Kombiniert werden. Der Fehlertest arbeitet wie später in den Messungen (siehe Kapitel 5.1.2.2) genauer zu sehen besonders zuverlässig, wenn sich wenig gutartige Peers in der AKM befinden. Genau entgegengesetzt verhält es sich mit dem Stichprobentest. Schon

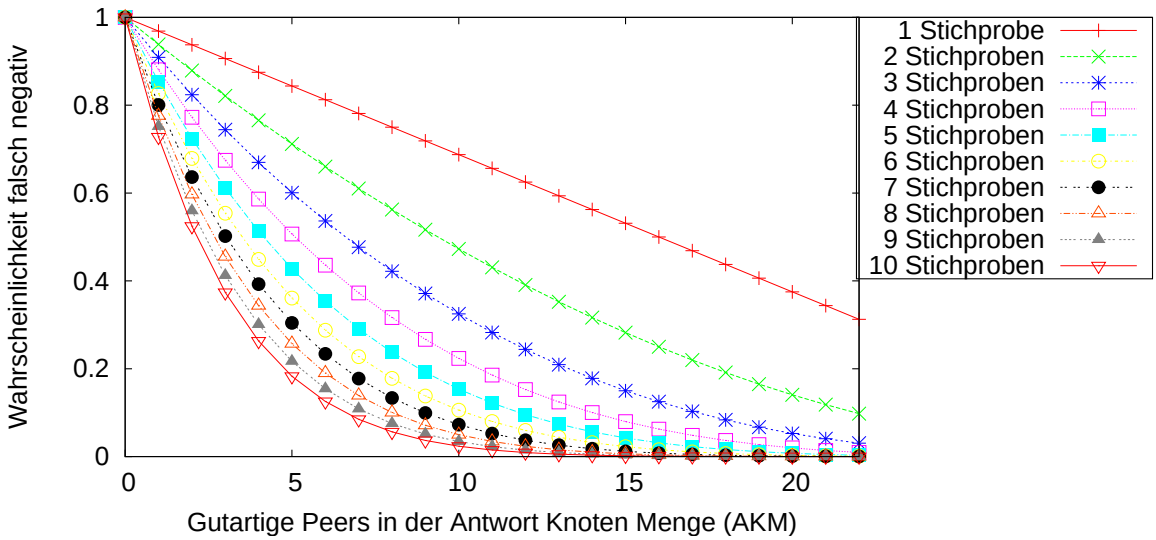


Abbildung 3.20: Falsch negativer Stichprobentest bei gemischter AKM

bei der Kombination von Fehlertest und lediglich vier Stichproben liegt die Wahrscheinlichkeit von falsch negativen Testergebnissen zuverlässig unter 2% (bei fünf Nachrichten sogar unter 1%). Im verlässlichen Chord wurde deshalb für die Überprüfung Chord interner Verwaltungsnachrichten der Stichprobentest mit dem Fehlertest kombiniert. Im Falle eines negativen Fehlertests wird der Stichprobentest für die AKM mit fünf Stichproben durchgeführt. Dies gilt, wenn die Antwort eines gutartigen Peers genügt. Insbesondere für die Feststellung, ob der antwortende Peer der Successor für die angeforderte $ChordID$ der Anfrage ist, ist dies der Fall. So kann überprüft werden, ob der richtige Peer geantwortet hat.

Redundantes Routing Durch die Fehlerkontrolle können Fehler beim Routing erkannt werden. Redundantes Routing hingegen dient dazu, diese Fehler zu umgehen. Das Vorgehen wurde von Castro u.a. [CDG+02] entwickelt und an Chord angepasst. Die Idee ist, eine Nachricht nicht nur entlang eines Pfades zum Zielknoten zu routen, sondern über viele Pfade. Im Prinzip handelt es sich deshalb um eine Flooding Abwandlung, allerdings unter Ausnutzung der Routinginfrastruktur von Chord. Werden genügend Nachrichten ins Netzwerk ausgesendet, erhalten mit großer Wahrscheinlichkeit alle Knoten im Netzwerk mindestens eine Nachricht.

Der Routingmechanismus von Chord muss für das redundante Routing insoweit angepasst werden, dass er nicht nur den Stammknoten für die gesuchte ChordID findet, sondern auch die Knoten, die Replikate mit dieser ChordID speichern. So können dann auch Datenanfragen durch redundantes Routing unterstützt werden.

Der redundante Routing Algorithmus liefert als Ergebnis eine Menge von Knoten mit den Knoten der Replikationsgruppe sowie den Stammknoten für die gesuchte ChordID zurück. Diese Menge wird als Sichere Knoten Menge $SKM(id)$ für die gesuchte ChordID id bezeichnet. Die SKM im verlässlichen Chord Protokoll enthält entsprechend 32 Knoten.

Für Chord stellt sich der redundante Routing Algorithmus wie folgt dar, falls Knoten n eine Nachricht mit ChordID req_id routen möchte:

1. n wählt unter seinen Einträgen der Fingertabelle die msg_hop_1 Einträge aus mit der geringsten Distanz zu req_id. An jeden Knoten der ausgewählten Fingertabelleneinträge wird eine FindNeighbour-Anfrage mit req_id weitergeleitet.

2. Jeder gutartige Peer, der eine solche Anfrage erhält, überprüft, ob er den Knoten $successor(req\_id)$ in seiner Replikationsgruppe hat. Ist dies der Fall, antwortet er an n. Ansonsten wird die Nachricht weitergeleitet. Hierzu werden wiederum die msg_hop_n Einträge mit der geringsten Distanz zu req_id ausgewählt und an diese Knoten die FindNeighbour-Anfrage weitergeleitet. msg_hop_n bestimmt sich hierbei nach der Anzahl von Routingschritten, die eine FindNeighbour-Nachricht schon absolviert hat. Bösartige Peers antworten in diesem Schritt nie und leiten die Nachricht auch nicht weiter, da sie keinerlei Interesse haben, dass n die korrekte Knotenmenge $SKM(id)$ findet.

3. n sammelt die Antworten auf die FindNeighbour-Anfragen ein und fügt 32 Peers mit der geringsten Distanz zu req_id zur Resultatmenge $\mathcal{M}$ hinzu. Nach einer Wartezeit auf weitere Antworten wird mit Schritt vier fortgefahren.

4. n sendet NodeList-Nachrichten mit der Resultatmenge $\mathcal{M}$ an alle Knoten von $\mathcal{M}$, die noch keine NodeList-Nachricht erhalten haben.

5. Gutartige Knoten vergleichen die Resultatmenge $\mathcal{M}$ mit der eigenen Replikationsgruppe. Sind darin noch Knoten enthalten, die nicht in $\mathcal{M}$ enthalten sind, werden diese in die Antwort der NodeList-Anfrage aufgenommen und an n zurückgesendet. Bösartige Knoten senden immer eine Liste der bösartigen Knoten mit minimaler Distanz zu req_id als Antwort zurück.

6. Die NodeList-Antworten werden von n gesammelt. Antwortet ein Knoten nicht, wird er aus der Resultatmenge $\mathcal{M}$ gelöscht. Befinden sich unter den neuen Knoten, die mit den NodeList-Antworten zurückgesendet wurden, Knoten mit geringerer Distanz als jene der Resultatmenge $\mathcal{M}$, oder ist die Menge $\mathcal{M}$ noch kleiner als ar (Anzahl der Replikate), werden von n an diese Forward-Nachrichten gesendet.

7. Knoten, die auf die Forward-Nachricht geantwortet haben, werden in die Resultatmenge $\mathcal{M}$ aufgenommen. Anschließend wird mit Schritt 4 fortgefahren.

8. Haben alle Knoten in $\mathcal{M}$ schon eine NodeList-Nachricht erhalten und darauf geantwortet, oder wurde Schritt 6 schon dreimal durchgeführt, wird der Algorithmus mit der Resultatmenge $\mathcal{M}$ beendet.

Die entscheidenden Punkte des Algorithmus sind die Punkte 1 und 2. Wird bei diesen zwei Punkten kein gutartiger Peer erreicht, der antwortet, bricht der Algorithmus ohne Ergebnis ab. Die Anzahl der msg_hop_1 bis msg_hop_n Weiterleitungen von Nachrichten ist hier die entscheidende Größe, um das Scheitern des Algorithmus zu verhindern. Entsprechende Simulationen und Messungen, um geeignete Größen zu ermitteln, werden in Kapitel 5 vorgestellt.

3.5.7.4 Gehärtete Verwaltung der Routinginformationen

Um die Einträge der Fingertabelle sowie der Predecessor- und Successor-Listen aktuell zu halten ist es nötig, diese regelmäßig zu überprüfen. Die Fingereinträge werden hierbei, wie auch beim reinen Chord Protokoll, der Reihe nach überprüft. Hierzu sendet der Knoten n

```
public NodeHandle closestPrecedingFinger(Id searchId) {
  //Laufe über die Einträge der Fingertabelle
  NodeHandle result = null;
  for (int i = finger.length - 1; i >= 0 ; i--) {
    //Eintrag existiert
    if (finger[i] != null){
      //finger[i].id liegt zwischen (id,searchId]
      if (finger[i].getId().betweenE(id, searchId)) {
        result = finger[i];
        break;
      }
    }
  }
  //Erweiterung: Verbessere das Ergebnis durch die Successor-Liste
  for (int i = 0;(i< routingSuccListMax &&
          !result.getId().equals(searchId));++i){
    Id altPrec = succList[i].getId();
    //succList[i].id liegt zwischen [result.id,searchId]
    if (altPrec.equals(searchId)){
      result = succList[i];
    }else if (altPrec.betweenE(result.getId(), searchId)){
      result = succList[i];
    }
  }
  return result;
}
```

Abbildung 3.21: Bestimmung des nächsten Predecessors aus der Fingertabelle

eine FindSucc-Nachricht mit $n.finger[k].id$ an den Predecessor mit der kürzesten Distanz zum aktuell gesuchten Fingereintrag, wobei k der Index des aktuell zu überprüfenden Fingereintrags darstellt. Der Predecessor der Fingertabelle mit der geringsten Distanz lässt sich anhand dem in Abbildung 3.21 gezeigten Pseudocode ermitteln. Die Nachricht wird gemäß dem Chord Protokoll zum $successor(n.finger[k].id)$ geroutet. Dieser antwortet, falls er gutartig ist, mit seiner Predecessor-Liste als AKM, denn nur die Predecessor-Knoten des antwortenden Peers a können beantworten, ob a tatsächlich $successor(n.finger[k].id)$ oder ob a nicht doch $bad\_successor(n.finger[k].id)$ ist. Ist a $bad\_successor(n.finger[k].id)$ gibt es andere Knoten, die eine geringere Distanz zu $n.finger[k].id$ aufweisen. Da in Chord im Uhrzeigersinn geroutet wird, speichern die Knoten nur Informationen zu nachfolgenden Knoten in ihren Fingertabellen. Deshalb haben nur Predecessor-Knoten von a Informationen über den korrekten $successor(n.finger[k].id)$ gespeichert.

n nimmt anhand der vom antwortenden Peer gelieferten Knotenmenge die Fehlerkontrolle mit Stichprobentest vor. Ist der Befund negativ, wird das zurückgelieferte Ergebnis akzeptiert – allerdings nur, falls der zurückgelieferte Knoten näher an $n.finger[k].id$ liegt als der momentane Eintrag. Dieses Vorgehen ist eine weitere Sicherheitsmaßnahme, um bösartigen Peers die Manipulation zusätzlicher Fingertabelleneinträge zu erschweren. Fingereinträge können nur einen Eintrag mit einem Knoten erhalten, der eine größere Distanz zu $n.finger[k].id$ aufweist, falls zuvor das Verschwinden des Knotens bemerkt und der Eintrag gelöscht wurde. Ist das Ergebnis des Fehlertests oder der anschließende Stichprobentest positiv, leitet der Knoten n

ein redundantes Routing für $n.finger[k].id$ ein. Aus der zurückgelieferten Knotenmenge wird der $successor(n.finger[k].id)$ bestimmt und der Fingertabelleneintrag $n.finger[k].knoten$ auf diesen Knoten gesetzt.

Besonders kritisch für die Funktion des Netzwerks und des Chord Routing Protokolls ist die korrekte Einordnung des Knotens in den bestehenden Chord Ring anhand seiner ChordID. Jeder Knoten n überprüft deshalb regelmäßig, ob der von ihm gespeicherte Successor s noch existiert und ob $s = successor(n.id)$. Durch neue, dem Netzwerk beigetretene Knoten kann sich der Successor des Peers ändern. Selbiges gilt für den Predecessor. Auch dieser wird regelmäßig vom Peer überprüft. Tritt ein Knoten dem Netzwerk bei, unterrichtet er die benachbarten Peers (insbesondere Successor und Predecessor) über seine Anwesenheit. Nur so wird er erfolgreich in den Chord Ring integriert. Allerdings können Peers das Netzwerk unvorhergesehen und damit ohne Benachrichtigung der benachbarten Peers verlassen (auch als „fail“ des Knotens bezeichnet). Um diese Situationen zu erkennen und die Funktion des Netzwerks auch dann zu garantieren, ist eine regelmäßige Überprüfung deshalb unabdingbar.

3.5.7.5 Beitritt von Knoten zum Netzwerk

Ähnlich wie beim unstrukturierten Netzwerk ist die Auswahl geeigneter Bootstrap-Knoten sehr kritisch. Die Bootstrap-Knoten werden deshalb auch hier von der Zertifizierungsinstanz erfragt. Für den Beitritt und damit die Einordnung des Knotens in das Chord Netzwerk wird wiederum auf das redundante Routing zurückgegriffen. Will Knoten n mit ChordID $n.id$ dem Chord Netzwerk beitreten, wird er von seinen Bootstrap-Knoten eine sichere Knoten Menge $SKM(n.id)$ anfordern. Sofern diese Knoten gutartig sind, werden sie ein redundantes Routing für $n.id$ initiieren und die so ermittelte Knotenmenge $SKM(n.id)$ an n zurückliefern. n wird die $SKM(n.id)$ der Bootstrap-Knoten sammeln und zu einer Menge zusammenführen. Aus dieser Resultatmenge werden die initialen Einträge für die Fingertabelle sowie die Successor-Liste erstellt. Durch dieses Vorgehen ist gewährleistet, dass beim Eintritt eines Peers ins Netzwerk der Anteil schlechter Fingertabelleneinträge dem Anteil bösartige Knoten im Netzwerk entspricht. Dies ist Voraussetzung, um eine korrekte Integration ins Chord Netzwerk auch bei bösartigen Knoten zu garantieren.

3.5.7.6 Optimierung des Chord Routings Protokolls

Wie bereits in Abbildung 3.21 dargestellt, wurde das Standard Chord Protokoll beim Suchen des nächsten Predecessors erweitert. Anstatt sich nur auf die Einträge der Fingertabelle zu verlassen werden durch die Erweiterung nun auch die Einträge der Successor-Liste hinzugezogen. Hierdurch kann wie später zu sehen, die Anzahl der nötigen Routingschritte reduziert und damit die Effizienz des Chord Protokolls erhöht werden.

3.5.7.7 Zusammenfassung der eigenen Beiträge zum verlässlichen Chord Protokoll

Das Standard Chord Protokoll wird an verschiedenen Stellen erweitert, um es gegenüber Angriffen von bösartigen Peers zu schützen. Der Beitrag des Autors hierzu liegt zum einen in der Übertragung der Ideen von Castro u.a. [CDG+02] und zum anderen in der Entwicklung weiterer Ideen, die das Chord Protokoll sicherer machen.

Erweiterungen des redundanten Routings Das redundante Routing, wie von Castro u.a. [CDG+02] beschrieben, wurde erweitert, sodass die Redundanz der Nachrichten über mehrere Routingschritte erzeugt werden kann. Im ursprünglichen Vorschlag wird die Anzahl der redundanten Nachrichten vollständig vom Peer erzeugt, der das redundante Routing der Nachrichten startet. Dies ist aufgrund der geringeren Anzahl an Fingereinträgen in PACS nicht möglich. Deshalb werden die Nachrichten wie beschrieben über mehrere Routingschritte redundant über mehrere Fingereinträge weitergeleitet.

Eigene Ideen zur Absicherung des Chord Protokolls Die Verwaltung der Fingereinträge wurde neu entwickelt. Die Bedingungen für die Fingereinträge sind schon durch das Chord Protokoll festgelegt, reichen aber, wie dargestellt, nicht aus. Nur durch die konsequente Anwendung redundanten Routings für die Suche neuer Fingereinträge kann die Manipulation dieser Einträge ausgeschlossen werden. Im Zuge dessen wurde die Predecessor-Liste in Chord eingeführt. Erst durch diese wird es möglich, die Fehlerkontrolle auch für die Suche nach neuen Fingereinträgen einzusetzen. Zudem wurden für das Verändern eines Fingereintrags zusätzliche Einschränkungen entwickelt, die ebenfalls die Manipulation der Fingertabelle erschweren. Zuletzt ist noch die Erweiterung des Standard Chord Routing Protokolls zu nennen. Diese reduziert die Anzahl der Routingschritte und damit auch die Angriffsstellen für Manipulationen.

3.5.7.8 DHT-Anwendung

Die DHT-Anwendung, die auf dem verlässlichen Chord Netzwerk aufbaut, wurde in PACS so erweitert, dass sie eine verlässliche Speicherung garantiert. Die DHT-Methoden ($put(key, priv)$, $get(key)$ $delete(key, priv)$, $put\_data(key, value)$, $get\_data(key)$ und $delete\_data(key)$) nutzen den Fehlertest und redundantes Routing, um die Operationen korrekt auszuführen bzw. deren Resultate zu überprüfen. Das Vorgehen ist hierbei auf der hier beschriebenen Abstraktionsebene für Datenobjekte wie für Berechtigungen dasselbe. Entsprechend wird bei den Ausführungen nicht zwischen Datenobjekt und Berechtigung unterschieden.

Einfügen von Datenobjekten Das Einfügen eines neuen Privilegs (Datenobjektes) geschieht durch die Methode $put(key, priv)$ ($put\_data(key, value)$).

Die Berechnung der ChordID des zu speichernden Datenobjekts wird durch konsistentes Hashing des *keys* der Berechtigung (Datenobjekts) und Abbildung in den Chord Schlüsselraum $ChordID(h(key)) = id\_key$ erzeugt. Anschließend sendet die DHT-Anwendung eine entsprechende Insert-Nachricht mit key, $value$ und id_key. Diese wird mithilfe des zugrundeliegenden Chord Netzwerks zum $successor(id\_key)$ geroutet. Dieser speichert das angegebene Datenobjekt $value$ unter dem Schlüssel key. Zusätzlich erzeugt er, da er der Stammknoten von id_key ist, Insert-Replika-Nachrichten, die an die Knoten seiner Replikationsgruppe gesendet werden. Diese speichern das Datenobjekt $value$ ebenfalls unter dem Schlüssel key. Das gilt allerdings nur, wenn der Stammknoten gutartig ist und die Nachricht nicht vorher durch bösartige Peers fehlgeleitet wird. Deshalb muss der $successor(id\_key)$ nach erfolgreicher Speicherung diese dem anfragenden Peer bestätigen. Dies geschieht durch die Beantwortung der Insert-Nachricht mit der Replikationsgruppe des Stammknotens als AKM. Anhand dieser AKM führt der Peer, der die Anfrage gestartet hat, den Fehlertest durch. Ist dieser negativ, startet die Anwendung einen Stichprobetest, indem sie Knoten der AKM nach dem zu speichernden Datenobjekt und der Successor- und Predecessor-Liste fragt. Hierbei sind

ebenfalls nur die fünf Stichproben nötig, da ein korrekter Peer ausreicht um festzustellen, ob die Daten wirklich repliziert wurden. Anhand der Successor- und Predecessor-Listen kann zudem überprüft werden, ob der $successor(id\_key)$ geantwortet hat. Haben alle Peers der Stichprobe das Datenobjekt gespeichert und der korrekte Peer geantwortet, ist das Einfügen des Datenobjekts erfolgreich beendet. Gab der Fehlertest oder die Stichprobe jedoch ein positives Ergebnis, erzeugt der Peer eine sichere Knoten Menge $SKM(id\_key)$, indem er das redundante Routing für id_key startet. Die Knoten der $SKM(id\_key)$ erhalten anschließend direkte Insert-Replika-Nachrichten zugestellt, die sie zur Speicherung des Datenobjekts $value$ unter dem Schlüssel key veranlassen.

Replikation der Datenobjekte Wann immer sich die Replikationsgruppe eine Peers verändert, müssen entsprechende Anpassungen an den Replikaten vorgenommen werden. Ist ein Knoten der Replikationsgruppe nicht mehr erreichbar und wird entsprechend aus dieser entfernt, wird der nun nächste ar Successor neu in die Replikationsgruppe aufgenommen. Er erhält die von diesem Knoten als Stammknoten verwalteten Replikate.

Kommt ein neuer Peer zur Replikationsgruppe von Knoten n hinzu, da dessen ChordID näher als der am weitesten entfernte Knoten von $n.rep$ an $n.id$ liegt, erhält dieser ebenfalls die von diesem Knoten als Stammknoten verwalteten Replikate.

Interessant ist die Situation, wenn der Stammknoten von Datenobjekten wechselt. Hierbei gilt es zwei Fälle zu unterscheiden.

Stammknoten verlässt das Netzwerk Wenn ein Knoten n das Netzwerk verlässt, wird $n.successor$ Stammknoten, der zuvor von n als Stammknoten verwalteten Datenobjekte. Da $n.successor$ Bestandteil von $n.rep$ ist, besitzt er sämtliche Datenobjekte von n und ist somit sofort imstande, diese Aufgabe auszuüben. $n.successor$ wird zudem an den am weitesten entfernten Knoten der $n.successor.rep$ Menge Replikate der von n neu als Stammknoten übernommenen Datenobjekte senden, um die Anzahl der im Netzwerk zu Verfügung stehenden Objekte wieder auf die festgelegte Anzahl zu erhöhen.

Beitritt eines Knotens zum Netzwerk Wird ein neuer Knoten a in das Netzwerk aufgenommen, dessen ChordID $a.id$ zwischen $n.predecessor$ und $n.id$ liegt, wechselt der Stammknoten ebenfalls. Der neue Knoten a wird für alle Datenobjekte, deren ChordIDs zwischen $n.predecessor.id$ und $a.id$ liegen, Stammknoten. Diese Datenobjekte werden deshalb von n an a übertragen.

Bösartige Stammknoten nehmen weder die Replikation der Daten, noch die Übertragung der Daten an einen neuen Stammknoten vor. Die Sicherstellung der Replikation bei einem bösartigen Stammknoten übernimmt der einfügende Peer durch die Überprüfung des Ergebnisses der Einfügeoperation und entsprechender direkter Insert-Nachrichten an die Peers der Replikationsgruppe der ermittelten SKM. Die Aufrechterhaltung der Replikation ist gefährdet, wenn Knoten der Replikationsgruppe wechseln, da der bösartige Stammknoten diesen keine Replikate gibt.

Neu ins Netzwerk hinzukommende Knoten fragen deshalb, sobald sie im Netzwerk vollständig integriert sind, ihre nächsten Successor-Knoten nach den von diesen gespeicherten Datenobjekten. Die Korrektheit der zurückgelieferten Datenobjekte muss hierbei vom Peer überprüft werden, um zu verhindern, dass sie ungültige Datenobjekte speichern, die sie von bösartigen Peers erhalten haben. So ist es dem neuen Knoten möglich, auch Replikate der

Datenobjekte zu erhalten, die einen bösartigen Stammknoten besitzen und entsprechende Aufgaben als neuer Stammknoten für diese wahrzunehmen.

Schon bestehende Knoten, die aufgrund des Weggangs eines Peers in die Replikationsgruppe eines bösartigen Peers aufgenommen werden, betreiben diesen Aufwand nicht. Die Gefahr, dass aufgrund dieses Verhaltens Datenobjekte vollständig aus dem Netzwerk verschwinden, ist sehr gering. Schließlich müssen hierzu alle Knoten der Replikationsgruppe nacheinander ausfallen, ohne dass in der Zwischenzeit ein neuer Knoten dem Netzwerk beitritt, der aufgrund seiner ChordID die Replikate anfordert. Der Ausfall des bösartigen Knoten selbst oder eine zwischenzeitliche Aktualisierung der Datenobjekte würden das Verschwinden ebenfalls verhindern.

Abfragen von Datenobjekten Die Abfrage von Datenobjekten geschieht durch die Methode $get\_data(key)$ bzw. $get(key)$. Die Berechnung der ChordID des abzufragenden Datenobjekts wird durch konsistentes Hashing des *keys* und Abbildung in den Chord Schlüsselraum $ChordID(h(key)) = id\_key$ erzeugt. Anschließend sendet die DHT-Anwendung eine entsprechende Lookup-Nachricht mit key und id_key. Die Nachricht wird wiederum gemäß dem Routingprotokoll von Chord zum $successor(key\_id)$ geroutet. Dieser antwortet mit dem unter dem Schlüssel key abgespeicherten Datenobjekten und seiner Replikationsgruppe als AKM. Der anfragende Knoten wird anhand der erhaltenen AKM und der erhaltenen Antwort die Überprüfung des Ergebnisses vornehmen. Hierzu führt er zunächst den Fehlertest mit der AKM durch. Ist der Fehlertest negativ, wird das Ergebnis weiter überprüft. Hierzu werden, wie beim Einfügen von Datenobjekten, zunächst als Stichprobe fünf Peers der AKM aufgefordert, die Datenobjekte, die sie unter dem key gespeichert haben sowie ihre Successor- und Predecessor-Liste zurückzuliefern. Anhand der Successor- und Predecessor-Liste wird überprüft, ob der korrekte Peer $successor(id\_key)$ geantwortet hat. Waren diese Prüfungen erfolgreich, erfolgt die inhaltliche Prüfung der erhaltenen Resultate. Hierzu dient die Multimenge SAM als Input, die aus den aus den Stichproben erhaltenen Datenobjekten und dem ursprünglichen Abfrageergebnis besteht.

Ist das Ergebnis des Fehlertests positiv oder die Überprüfung ergab, dass nicht der korrekte Successor geantwortet hat, wird eine $SKM(id\_key)$ durch redundantes Routing für id_key angefordert. Aus dieser $SKM(id\_key)$ werden durch direkte Anfragen ebenfalls fünf Stichproben ermittelt. Diese fünf Stichprobenergebnisse stellen dann die neue Multimenge SAM, die als Input für die inhaltliche Prüfung dient.

Die inhaltliche Überprüfung unterscheidet sich je nach Art der angefragten Daten. Wurden Berechtigungen durch $get(key)$ angefragt, folgt eine Überprüfung der zurückgegebenen Antwort anhand der in Kapitel 3.5.4.3 vorgestellten Techniken zur Erkennung ungültiger Berechtigungen (Stichprobenprüfung oder durch Überprüfung des Delegationspfades). Bei beiden Prüfungsarten werden die durch die erste Stichprobe gewonnenen Berechtigungen SAM mit einbezogen.

Als Ergebnis werden nur Berechtigungen zurückgegeben, die gemäß dieser Überprüfung als korrekt akzeptiert wurden. Bei Datenobjekte die durch $get\_data(id\_key)$ Aufrufe angefragt werden, erfolgt für die zurückerhaltenen Datenobjekte in der SAM ebenfalls eine inhaltlich Prüfung. Diese Prüfung ist aber abhängig von den in diesen Datenobjekten gespeicherten Daten. Auch hier werden nur Ergebnisse zurückgegeben, die diese inhaltliche Überprüfung bestehen.

3.5.8 Abfragearten von Berechtigungen

Ausschlaggebend für die nachfolgende Organisation bzw. den Aufbau des Privilegienspeichers sind die Kenntnis und die Bewertung der von ihm zu unterstützenden Anfragen nach Privilegien und Benutzerzuordnungen. Diese Anfragen werden durch die Privilegienverwaltung und bei Datenanfragen durch die Durchsetzungskomponente gestellt.

Anfragen an den Privilegienspeicher sind üblicherweise wie folgt. Ermittle

1. die Privilegien von Benutzer b an Datenobjekt do mit Aktion a (und lokalem Benutzer lb)
2. die Privilegien der Rolle r an Datenobjekt do mit Aktion a (und lokalem Benutzer lb)
3. die Benutzerzuordnungen von Benutzer b für Rolle r
4. welcher Benutzer ein Privileg von Benutzer b an Datenobjekt do mit Aktion a und lokalem Benutzer lb erhalten hat
5. welche Rolle ein Privileg von Benutzer b an Datenobjekt do mit Aktion a und lokalem Benutzer lb erhalten hat
6. welcher Benutzer eine Benutzerzuordnung von Benutzer b für Rolle r erhalten hat
7. alle Benutzer einer Rolle b
8. alle administrativen Privilegien einer Rolle b
9. alle Benutzer und Rollen, die ein Privileg an Datenobjekt do mit Aktion a (und lokalem Benutzer lb) besitzen
10. alle Privilegien des Benutzers b
11. alle Rollen eines Benutzers b
12. alle Privilegien einer Rolle b

Abfrage eins und zwei dienen der Überprüfung eines Privilegs. Sie werden deshalb bei jeder Anfrage eines Subjekts benötigt. Es ist dabei unerheblich, ob es sich um die Anfrage eines Datenobjekts oder um die Aufforderung zur Erteilung eines Privilegs oder einer Benutzerzuordnung handelt. Bezieht der Benutzer seine Privilegien über eine Rolle, ist diese durch Abfrage drei zu überprüfen. Die Abfragen eins bis drei werden deshalb am häufigsten verwendet.

Die Abfragen vier bis acht dienen alle der effizienten Durchführung des kaskadierenden Widerrufs von Privilegien und Benutzerzuordnungen. Abfrage vier und fünf werden für das kaskadierende Widerrufen eines administrativen Privilegs benötigt. Abfrage sechs dient dem kaskadierenden Widerruf einer Benutzerzuordnung, falls die Benutzerzuordnung mit Grantoption gewährt wurde. Hierzu ist es nötig, die Benutzerzuordnungen zu ermitteln, die auf der zu löschenden Benutzerzuordnung basieren. Abfrage acht wird für das Widerrufen einer Benutzerzuordnung benötigt. Hierzu ist das kaskadierende Entziehen aller administrativen Privilegien der Rolle nötig, weshalb alle administrativen Privilegien der Rolle ermittelt werden müssen. Anfrage sieben wird benötigt, um beim Widerruf eines administrativen Privilegs einer Rolle alle Benutzer ermitteln zu können, für die ein kaskadierendes Widerrufen dieses

Privilegs vorgenommen werden muss. Die Abfragen fünf bis acht müssen zwar unterstützt werden, kommen aber wesentlich seltener zum Einsatz als Abfragen eins bis drei.

Abfragen neun bis zwölf dienen lediglich der Kontrolle des Dateneigentümers (um festzustellen, wer Zugriff auf ein Datenobjekt besitzt) und des Benutzers (um festzustellen, welche Privilegien und Rollen er besitzt). Diese Anfragen sind entsprechend optional und müssen durch den Privilegienspeicher nicht unterstützt werden.

3.5.9 Zusammenfassung

Bei der Gegenüberstellung von unstrukturierten Privilegienspeichern mit definierter Replikationsgruppe (im Folgenden unstrukturierte P2P-Privilegienspeicher genannt) und DHT-Privilegienspeicher, wird die wesentlich aufwändigere Verwaltung des DHT-Privilegienspeichers deutlich. Die Verwaltungsstrukturen des dort verwendeten Chord Netzwerks bieten Angriffspunkte für bösartige Knoten und müssen deshalb durch die vorgestellten Erweiterungen abgesichert werden. Dieser Verwaltungsaufwand steigt mit der Fluktuation von Netzwerkteilnehmern, denn jeder hinzukommende und weggehende Knoten verursacht Änderungen an den Fingertabellen, sowie an den Successor- und Predecessor-Listen der Netzwerkknoten.

Beim unstrukturierten P2P-Privilegienspeicher sind die Auswirkungen der Fluktuation von Netzwerkteilnehmern wesentlich geringer, da dort weniger Verwaltungsinformationen verwaltet werden. Allerdings kommt hier noch die Verwaltung der Replikationsgruppe hinzu, deren Verwaltungsaufwand mit der Anzahl der Änderungen im Netzwerk ansteigt. Trotzdem ist der unstrukturierte P2P-Privilegienspeicher insgesamt gegenüber Änderungen im Netzwerk robuster, als der DHT-Privilegienspeicher.

Der Zugriff auf den Privilegienspeicher erfolgt bislang durch die Methode $get(key)$. Der DHT-Privilegienspeicher ist zwar bei der Suche nach Privilegien aufgrund der strukturierten Ablage der Datenobjekte effizienter, allerdings wird nur die Suche nach einem Schlüssel unterstützt. Für die präsentierten Abfragearten der Berechtigungen, ist die Abfragemöglichkeit über einen Schlüssel jedoch nicht ausreichend. Ist dieser aber ganz oder auch nur teilweise unbekannt, wie zum Beispiel bei Bereichsabfragen, kann dieser Suchmechanismus nicht genutzt werden. Deshalb sollte der Schlüssel so gewählt werden, dass zumindest die häufigsten Anfragen, also (1-3), unterstützen werden können.

Diese Beschränkung der DHT macht den Vorteil der Speicherung im unstrukturierten Netzwerk sichtbar. Der Schlüssel dient dort nur der Identifikation des Privilegs und ist deshalb für die Abfrage der Privilegien unbedeutend. Durch die Flooding-Technik können beliebig komplexe Abfragen an die Peers geschickt werden, die diese dann auswerten und entsprechend beantworten. Diesem Vorteil steht allerdings die wesentliche aufwändigere Suche nach gespeicherten Informationen gegenüber. Denn hierbei müssen alle Knoten des Netzwerk kontaktiert werden.

Sowohl der DHT-Privilegienspeicher als auch der unstrukturierte P2P-Privilegienspeicher garantieren, trotz ihres ganz unterschiedlichen Vorgehens, die zuverlässige Speicherung der Datenobjekte. Die Entscheidung, welcher von beiden Alternativen nun der effizientere Privilegienspeicher ist, ist vor allem davon abhängig, wie die Privilegien in diesem Privilegienspeicher abgelegt werden. Zudem ist entscheidend, wie oft auf die im Privilegienspeicher abgelegten Privilegien bei der Durchsetzung der Zugriffskontrolle zugegriffen werden muss. Eine allgemein gültige Empfehlung kann deshalb an dieser Stelle nicht gegeben werden.

3.6 Ablauf einer Datenanfrage in PACS

Zum besseren Verständnis der Durchsetzung der Zugriffskontrolle in den Kapiteln 3.7 und 3.8 wird in diesem Kapitel zunächst der prinzipielle Ablauf einer Datenanfrage in PACS dargestellt. Bei jeder Datenanfrage läuft die Auswertung der Zugriffskontrolle ab. Deren Resultat bestimmt das Verhalten der Durchsetzung der Zugriffskontrolle gegenüber dem Anfragenden, das dann in den Folgekapiteln beschrieben wird.

Die Zugriffskontrolle in PACS obliegt denselben Grundsätzen wie in einem zentralen System. Nach der Authentifizierung des Subjekts wird für jede Anfrage eines Datenobjekts die Autorisierung des anfragenden Subjekts für die angeforderte Aktion überprüft (Durchsetzung der Zugriffskontrolle). Der Ablauf dieser Überprüfung wird in diesem Kapitel dargestellt.

Bestandteile der Autorisierungsprüfung sind:

- Das zu autorisierende Subjekt
- Die aktivierten Rollen des Subjekts
- Der Benutzer, zu dem das Subjekt gehört
- Das Objekt, auf das sich die Autorisierung bezieht
- Die Aktion, die auf diesem Objekt durchgeführt werden soll

Optional können noch folgende Angaben vom Subjekt geliefert werden, um den Prozess der Autorisierung zu beschleunigen.

- Der lokale Benutzer, der die Aktion genauer spezifiziert
- Eine Menge von Privilegien, die das Subjekt autorisiert, die angeforderte Aktion auf dem Objekt durchzuführen

Die Bereitstellung der Privilegien durch das Subjekt während der Anfrage ist ein übliches Vorgehen bei verteilten Systemen und reduziert den Kommunikationsaufwand zum Überprüfen der Autorisierung.

Die Überprüfung der Zugriffsrechte erfolgt durch den Referenzmonitor im PACS-Modul. Die Überprüfung einer Anfrage erfolgt gemäß nachstehender Abfolge:

1. Überprüfung der globalen Privilegien, fakultativ mit Überprüfung der Delegationspfade.
2. Evaluation der Export Policy.

Wurden beide Überprüfungen bestanden, ist die Anfrage erfolgreich autorisiert. Die einzelnen Teile der Überprüfung werden nun genauer erläutert.

3.6.1 Überprüfung der globalen Autorisierung

Zur Überprüfung der globalen Autorisierung besorgt sich der Referenzmonitor entweder die nötigen Privilegien und Benutzerzuordnungen aus dem Privilegienspeicher oder er erhält diese Informationen vom anfragenden Subjekt, nachfolgend Anfragender genannt. Erhält er diese Informationen vom Anfragenden, sind sie auf ihre Aktualität hin zu prüfen. Hierzu fragt der Referenzmonitor die vom Anfragenden erhaltenen Daten nochmals beim Privilegienspeicher

ab. Liegt eine Übereinstimmung vor, werden die Privilegien weiter überprüft, ansonsten wird die Anfrage abgewiesen.

Zur weiteren Überprüfung wird zunächst die Funktion $autorisiert(s, a, lb, o)$ mit s als Subjekt des Anfragenden, die angeforderte Aktion a, der lokalem Benutzer lb und das angefragte Objekt o ausgewertet. Die Mengen von PZ, BZ und PB werden hierbei auf die vom Anfragenden gelieferten Daten beschränkt. Gibt diese Funktion w zurück, war die Überprüfung der Autorisierung erfolgreich, ansonsten wird die Anfrage verworfen. Hat der Anfragende keine Privilegien mit der Anfrage geliefert, entfällt diese Optimierung.

3.6.2 Überprüfung des Delegationspfads

Die Überprüfung des Delegationspfades kann als Zusatzprüfung der globalen Berechtigungen betrachtet werden. Sollte es an der Korrektheit der globalen Berechtigungen Zweifel geben, kann der Delegationspfad der vom Anfragenden gelieferten Berechtigung überprüft werden. Dies geschieht durch die schon definierten Funktionen *check_zugr* bzw. *check_bz*.

3.6.3 Evaluation der Export Policy

Die Evaluation der Export Policy erfolgt nachdem die Überprüfung der globalen Privilegien erfolgreich abgeschlossen ist. Die dort festgelegte Aktion a, der lokale Benutzer lb und das Objekt o sind die nötigen Angaben, um die Evaluation zu starten. Für die Evaluation vertritt der lokale Benutzer den anfragenden Benutzer. Hierdurch wird eine nötige Änderung der Export Policy umgangen. Das Resultat der Evaluation wird wiederum dem Anfragenden zugeschrieben. Die Evaluation ist nötig, da in der Export Policy eventuell noch zusätzliche Restriktionen spezifiziert wurden, die im globalen Zugriffskontrollmodell nicht möglich sind. Die Evaluation dient der Durchsetzung dieser zusätzlichen Bedingungen.

War die Evaluation der Export Policy erfolgreich, wird die Anfrage an die lokale Datenbasis weitergeleitet, die diese dann beantwortet. Die Ausführung der Anfrage erfolgt hierbei unter dem lokalen Benutzer lb.

3.7 Serverseitige Durchsetzung der Privilegien

Die serverseitige Durchsetzung bietet sich als einfachste Form der Durchsetzung der Privilegien an. Für P2P-Systeme hat sie, aufgrund des beschränkten Vertrauens zwischen den Teilnehmern, den Nachteil, dass die Unterstützung von Datenreplikation nicht möglich ist (siehe Kapitel 2.5.1). Die Realisierung der serverseitigen Durchsetzung in PACS soll hier kurz vorgestellt werden.

3.7.1 Durchsetzung der Privilegien

Die Auswertung der Privilegien erfolgt bei einer Datenanfrage durch den Peer, der das Datenobjekt speichert. Da keine Replikation unterstützt wird, ist er identisch mit dem Dateneigentümer. Die Auswertung der Privilegien für eine Anfrage erfolgt hierbei entsprechend dem in Kapitel 3.6 erläuterten Vorgehen. Ist gemäß dieser Auswertung der Zugriff auf das angeforderte Datenobjekt autorisiert, wird die Anfrage vom Peer akzeptiert und entsprechend beantwortet. Vor der Übermittlung der Ergebnisse werden diese mit dem öffentlichen

Schlüssel des anfragenden Benutzers verschlüsselt, um zu verhindern, dass bei der Datenübertragung unautorisierte Dritte die Daten mitlesen können. Ergibt die Überprüfung, dass der Zugriff nicht autorisiert ist, wird die Anfrage nicht beantwortet. Durch die serverseitige Auswertung der Privilegien kann der gesamte Funktionsumfang von XACML in den Export Policies ausgenutzt werden.

3.7.2 Organisation des Privilegienspeichers bei serverseitiger Durchsetzung

Die Organisation der Berechtigungen im Privilegienspeicher besitzt zwei Dimensionen. Zunächst wird geprüft, welche Privilegien überhaupt im Privilegienspeicher abgelegt werden müssen. Anschließend kann dann über dessen optimalen Aufbau entschieden werden.

3.7.2.1 Speicherorte der Berechtigungen

Da keine Replikation der Daten stattfindet, müssen auch deren Privilegien nicht repliziert werden. Ist der Peer und damit seine Daten nicht im Netzwerk verfügbar, sind auch die Privilegien für diese Daten wertlos. Da die Durchsetzung der Privilegien immer beim Dateneigentümer stattfindet, bietet es sich an, alle Privilegien zu diesen Daten auch beim Dateneigentümer (dem Server) zu speichern und zu verwalten. Die durch administrative Delegation von anderen Peers vergebenen Privilegien werden ebenfalls beim Dateneigentümer abgelegt. Der Dateneigentümer hat dadurch eine umfassende Kontrolle über die vergebenen Privilegien an seinen Datenobjekten.

Privilegien an Datenobjekten können so nur vergeben und entzogen werden, wenn der Dateneigentümer verfügbar ist, da sie sonst nicht abgespeichert werden können. Auf die betroffenen Datenobjekte kann aber ohnehin erst zugegriffen werden, wenn der Dateneigentümer wieder im Netzwerk ist. Dann können auch die Privilegien gewährt bzw. entzogen werden.

Auch für die serverseitige Durchsetzung ist ein allgemeiner Privilegienspeicher nötig. Darin werden die folgenden Benutzerzuordnungen und Privilegien abgelegt:

Benutzerzuordnungen Die ausschließliche lokale Speicherung beim Dateneigentümer unterstützt die administrative Delegation von Benutzerzuordnungen nicht. Bei einer administrativen Delegation der Benutzerzuordnung muss die Rollenzugehörigkeit eines Benutzer auch bei Abwesenheit des Rolleneigentümers verlässlich überprüft werden können. Dies soll anhand der in Kapitel 3.1.4 vorgestellten Beispielanwendung verdeutlicht werden.

ernst@ethz hat *ula@uzh* die Rolle *ethz.r*1 mit Grantoption zugewiesen. *ula@uzh* gibt diese Rolle an *bastian@unibas* weiter. Das Problem ist nun, den optimalen Speicherort dieser Benutzerzuordnung zu bestimmen. Wird sie beim Eigentümer der Rolle (ETHZ) gespeichert, geht sie mit dem Verschwinden von ETHZ verloren. Wird sie beim Grantor (UZH) gespeichert, kann die Korrektheit dieser Zuordnung nicht überprüft werden, da die Zuordnung, die *ernst@ethz ula@uzh* gewährt hat, bei Abwesenheit von ETHZ nicht überprüft werden kann, da diese nur von ETHZ gespeichert wurde. Eine zusätzliche Speicherung der Zuordnung beim Grantee (UniBas) behebt das Problem ebenfalls nicht, da das Privileg eventuell vom Grantor (UZH) schon widerrufen wurde.

Durch die Speicherung der Benutzerzuordnungen im Privilegienspeicher wird dieses Problem behoben, da diese dort unabhängig von Grantee und Dateneigentümer speichert werden.

Privilegien von Rollen Die Ermittlung aller administrativen Privilegien einer Rolle ist ebenfalls durch den Privilegienspeicher zu lösen (Abfrage acht). Die Privilegien sind zwar beim Datenobjekt gespeichert, allerdings ist es nur durch Flooding des gesamten Netzwerks möglich, alle administrativen Privilegien einer Rolle zu ermitteln. Deshalb werden auch die einer Rolle zugewiesenen administrativen Privilegien im Privilegienspeicher abgelegt.

Export Policies Da die Benutzerzuordnungen letztlich auf der Export Policy des Peers fußen, ist es nötig, dass diese auffindbar bleibt, falls der Peer das Netzwerk kurz verlässt. Nur so ist sicherzustellen, dass die auf dieser Export Policy basierenden globalen Privilegien nach wie vor überprüft werden können.

DGIO Die Verwaltungsinformationen der Replikationsgruppen der Peers im Netzwerk müssen ebenfalls im Privilegienspeicher abgelegt werden. Dies ist die einzige Möglichkeit, die Integrität der DGIOs sicherzustellen. Die Speicherung der DGIOs ist natürlich nur erforderlich, falls der Privilegienspeicher für das unstrukturierte P2P-Netzwerk zum Einsatz kommt.

3.7.2.2 Speicherung im DHT-Privilegienspeicher

Bei der Speicherung der zuvor festgelegten Berechtigungen im DHT basierten Privilegienspeicher ist insbesondere die Wahl des Schlüssels für die Ablage von Daten im DHT-Speicher ausschlaggebend. Die Entscheidungen für die Schlüssel der im Privilegienspeicher abzulegenden Daten werden im Folgenden erläutert.

Speicherung der Benutzerzuordnungen Da alle Abfragen für die Benutzerzuordnungen unterstützt werden müssen, wurde der Rollenbezeichner als Schlüssel für die Benutzerzuordnungen bestimmt. Bei den Benutzerzuordnungsanfragen wird dann noch die genaue Abfrage mitgeliefert, sodass die Knoten, die diese speichern, die Ergebnisse entsprechend filtern können. So werden nur die für den anfragenden Peer relevanten Benutzerzuordnungen zurückgeliefert.

Speicherung der administrativen Privilegien einer Rolle Die Speicherung der administrativen Privilegien einer Rolle im Privilegienspeicher dient lediglich der leichteren Auffindbarkeit. Die Primärkopie dieser Privilegien liegt dabei stets beim Datenobjekteigentümer. Der Eintrag im Privilegienspeicher erleichtert es dem Benutzer, der eine Benutzerzuordnung entzieht, die Dateneigentümer ausfindig zu machen, die aufgefordert werden müssen, ein kaskadierendes Entziehen der Privilegien durchzuführen, die dieser Benutzer potenziell vergeben hat. Die Speicherung der administrativen Privilegien im Privilegienspeicher erfolgt hierbei durch den Eigentümer des Datenobjekts. Dieser hat ein eigenes Interesse, über relevante Änderungen an Benutzerzuordnungen unterrichtet zu werden, und stellt sicher, dass Änderungen an administrativen Privilegien von Rollen auch in den Privilegienspeicher übertragen werden.

Als Schlüssel für die administrativen Privilegien, die an Rollen vergeben wurden, wird ebenfalls der Rollenbezeichner gewählt. Nur so kann die Abfrage nach allen administrativen Privilegien einer Rolle durch die DHT unterstützt werden. Die Speicherung und Löschung derselben im Privilegienspeicher erfolgt im Zuge ihrer Speicherung und Löschung beim Datenobjekteigentümer. Der Datenobjekteigentümer vollzieht die Änderungen für den Privilegienspeicher nach.

Speicherung der Export Policies Export Policies werden wie in Kapitel 3.5.4.5 erläutert im Privilegienspeicher abgelegt und vom Privilegienspeicher entsprechend verwaltet. Hierdurch ist deren Schutz garantiert.

3.7.2.3 Abfragen der Daten im DHT-Privilegienspeicher

Da nur wenige Berechtigungen im Privilegienspeicher abgelegt werden, ist es möglich, alle relevanten Anfragen mittels geschickter Wahl der Schlüssel durch die DHT-Routing-Infrastruktur zu unterstützen. Auch die Überprüfung der so erhaltenen Privilegien wird möglichst effizient und ohne Flooding ermöglicht. Das Vorgehen ist hierbei wie folgt.

Abfrage der Benutzerzuordnungen Die vom Privilegienspeicher zurückgegebenen Benutzerzuordnungen müssen das Vorhandensein einer Benutzerzuordnung zweifelsfrei ermitteln können. Deshalb ist es für den Privilegienspeicher nach den in Kapitel 3.5.4.3 vorgestellten Verfahren zur Erkennung ungültiger Privilegien nötig, entweder alle 32 Proben auszuwerten oder aber den Delegationspfad der Berechtigungen zu überprüfen. Da alle Benutzerzuordnungen einer Rolle beim gleichen Peer gespeichert sind, ist die Auswertung des Delegationspfades die effizientere Alternative, da sie zumeist ohne Anfragen an andere Peers des Netzwerks auskommt. Lediglich die Export Policy des Rolleneigentümers muss eventuell aus dem Privilegienspeicher abgefragt werden. Dies ist ein weiterer Vorteil der Speicherung der Benutzerzuordnungen unter dem Rollenbezeichner.

Abfrage der administrativen Privilegien für Rollen Bei der Abfrage dieser administrativen Privilegien gilt dasselbe Vorgehen wie bei den Benutzerzuordnungen. Allerdings liegen dem Abfragenden die nötigen Privilegien nicht lokal vor (sonst würde er diese ja nicht im Privilegienspeicher abfragen). Hier ist die Auswertung der Stichproben der Replikationsgruppe die effizientere Alternative.

Abfrage der Export Policy Bei der Abfrage der Export Policy stellt der Privilegienspeicher sicher, dass die aktuellste signierte Export Policy zurückgegeben wird. Da jeder Peer seine Export Policy im Privilegienspeicher ablegt, reicht der auch für den Fehlertest durchgeführte Stichprobentest mit fünf Stichproben aus. Es wird lediglich eine Antwort eines guten Peers benötigt, um die aktuelle Export Policy (die Export Policy mit der höchsten Versionsnummer unter denen mit gültiger Signatur) ermitteln zu können.

3.7.2.4 Speicherung im unstrukturierten P2P-Privilegienspeicher

Bei der Wahl des unstrukturierten Netzwerks mit definierter Replikationsgruppe besteht der Schlüssel beim Speichern der Berechtigung aus deren Primärschlüssel (siehe Kapitel 3.5.1). Bei $get(key)$ Anfragen besteht der key dann aus der konkreten Anfrage (also z.B. ermittle alle Benutzerzuordnungen von Benutzer b an Rolle r). Der key wird für das Routing nicht gebraucht. Eine direkte Abfrage nach dem key ist über die Abfrage des Primärschlüssels möglich.

Alle Abfragen werden durch Flooding im Netzwerk ermittelt, was sehr aufwändig ist. Hinzu kommt, dass aufgrund der beliebigen Festlegung der Replikationsgruppe der Stichprobentest nur unter bestimmten Bedingungen verwendet werden kann. Die Speicherung der Daten im Privilegienspeicher erfolgt deshalb in der nun vorgestellten Art und Weise.

Speicherung der Benutzerzuordnungen In PACS werden die Benutzerzuordnungen nicht beim Grantor (und dessen Replikationsgruppe) abgelegt, sondern beim Rolleneigentümer (und dessen Replikationsgruppe).

Die Replikationsgruppe des Rolleneigentümers, insbesondere deren Mitglieder, ist im DGIO des Rolleneigentümers definiert. Ist der Rolleneigentümer im Netzwerk, wird entsprechend die Speicherung der Benutzerzuordnung bei ihm direkt veranlasst. Dieser übernimmt dann die Replikation dieser Benutzerzuordnung auf die Peers seiner Replikationsgruppe.

Ist der Rolleneigentümer nicht im Netzwerk erreichbar, wird die Speicherung der Benutzerzuordnung direkt auf den Mitgliedern der Replikationsgruppe des Rolleneigentümers veranlasst. Hierzu muss der Grantor der Benutzerzuordnung das DGIO des Rolleneigentümers vom Privilegienspeicher abfragen. Sodann sendet der Grantor eine direkte Anfragen an die im DGIO definierten Mitglieder der Replikationsgruppe. Diese werden die Benutzerzuordnung speichern, sofern sie korrekt ist und es sich um einen gutartigen Peer handelt. Die Überprüfung des Delegationspfades des zu speichernden Privilegs ist hier für den Peer leicht möglich, da ihm sämtliche hierfür relevanten Benutzerzuordnungen lokal vorliegen.

Speicherung der administrativen Privilegien einer Rolle Die administrativen Privilegien einer Rollen werden bei dieser Lösung vom Datenobjekteigentümer an den Rolleneigentümer weitergegeben und in seiner Replikationsgruppe repliziert. Entsprechend werden Änderungen an den betroffenen administrativen Privilegien an den Rolleneigentümer bzw. an seine Replikationsgruppe übertragen.

Speicherung der Export Policies Export Policies werden analog zu den Ausführungen für die Speicherung der Export Policies im DHT-Privilegienspeicher gespeichert. Die Replikation dieser Export Policies erfolgt hierbei in der Replikationsgruppe des Peers, zu dem die Export Policy gehört.

Speicherung der DGIOs Jeder Peer ist für die Speicherung des DGIO seiner Replikationsgruppe im Netzwerk verantwortlich. Die Speicherung erfolgt unter dem systemweit festgelegten Schlüssel ID_{DGIO} (siehe Kapitel 3.8.3.2) in der Replikationsgruppe des Peers. Der Privilegienspeicher stellt hierbei die Integrität der abgelegten DGIOs sicher.

3.7.2.5 Abfragen im unstrukturierten P2P-Privilegienspeicher

Die Abfrage von Berechtigungen basiert bei unstrukturierten Netzwerken prinzipiell auf der Flooding-Technik. Durch die differenzierte Ablage von Berechtigungen im Privilegienspeicher wird versucht, die nötigen Flooding Anfragen auf ein Minimum zu reduzieren. Das Vorgehen für die einzelnen Berechtigungen ist hierbei wie folgt.

Abfrage der Benutzerzuordnung Bei der Abfrage von Benutzerzuordnungen kann ausgenutzt werden, dass die Replikationsgruppe von einer vertrauensvollen Autorität (dem Rolleneigentümer) erstellt wurde. Deshalb kann dort der Stichprobentest für Benutzerzuordnungen die Rollen des Rolleneigentümers betreffen angewandt werden. Zudem kann bei Benutzerzuordnungsanfragen auf das Flooding verzichtet werden. Diese haben ja nun einen durch das DGIO des Rolleneigentümers definierten Speicherplatz. Allerdings ist das DGIO des Rolleneigentümers zuvor durch Flooding zu erfragen.

Müssen die Mitglieder der Replikationsgruppe des Rolleneigentümers angefragt werden, ist es erforderlich, die Ergebnisse dieser Anfragen zu prüfen. Dazu werden die in Kapitel 3.5.4.3 vorgestellten Verfahren zur Erkennung ungültiger Privilegien angewendet. Durch die vertrauensvolle Replikationsgruppe des Rolleneigentümers können hierzu entweder alle 32 Proben ausgewertet oder der Delegationspfad der Privilegien überprüft werden. Da alle Benutzerzuordnungen einer Rolle beim gleichen Peer gespeichert sind, ist die Auswertung des Delegationspfades die effizientere Alternative. Auch die Export Policy des Peers wird durch seine Replikationsgruppe gespeichert, sodass auch diese ohne Flooding des Netzwerks erfragt werden kann, falls das DGIO vorliegt.

Abfrage administrativer Privilegien von Rollen Bei Abfrage administrativer Privilegien von Rollen ist dasselbe Vorgehen zu wählen wie bei Abfragen von Benutzerzuordnungen. Ist der Rolleneigentümer im Netzwerk erreichbar, wird dieser direkt angefragt; auf eine weitere Überprüfung des Resultats kann dann verzichtet werden.

Werden die Mitglieder der Replikationsgruppe des Rolleneigentümers konsultiert, müssen deren Ergebnisse gemäß denselben Kriterien überprüft werden, wie bei Abfragen von Benutzerzuordnungen. Die zur Überprüfung des Delegationspfades nötigen Privilegien liegen hierbei dem Anfragenden zumeist nicht lokal vor. Deshalb ist die Überprüfung des Delegationspfades nur selten die effizientere Alternative im Vergleich zur Auswertung der Stichproben der Replikationsgruppe.

Abfrage von Export Policies und DGIOs Abfragen dieser Art erfolgen via Flooding. Überprüfungen von Export Policies erfolgen analog dem Vorgehen beim DHT-Privilegienspeicher. Selbiges gilt für DGIOs. Da jeder Peer ein DGIO besitzt, reicht bei einer Abfrage eines DGIO Objekts der Stichprobentest mit fünf Stichproben aus. Es wird lediglich die Antwort eines guten Peers benötigt, um die anderen Antworten als falsch zu erkennen. Im Fall des DGIO ist die korrekte Antwort aus dieser Stichprobe immer das korrekt signierte DGIO mit der höchsten Versionsummern V_{DGIO} (siehe Kapitel 3.8.3.2).

3.7.3 Bedeutung der serverseitigen Durchsetzung

Da Privilegien meist am Ort der Auswertung lokal vorliegen, ist ihre serverseitige Umsetzung sehr effizient. Dies gilt insbesondere für Anfragen von Datenobjekten und die Speicherung von Privilegien. Lediglich für die Auswertung und Speicherung der Benutzerzuordnungen und sonstiger Verwaltungsdaten wie Export Policies und DGIOs, ist der Zugriff auf den Privilegienspeicher nötig.

Wird auf administrative Delegation von Benutzerzuordnungen verzichtet, kommt diese Durchsetzungsvariante sogar gänzlich ohne Privilegienspeicher aus. Dadurch gestaltet sich die Auswertung und damit die Durchsetzung der Privilegien noch effizienter, da der Verwaltungsaufwand des allgemeinen Privilegienspeichers entfällt.

Auch ohne diese zusätzliche Beschränkung ist der Aufwand, den PACS bei der serverseitigen Durchsetzung verursacht, sehr gering. Allerdings wird die sichere Replikation der Daten nicht unterstützt. Für Anwendungen wie PDMS und den Austausch von Daten, die nicht ständig verfügbar sein müssen, ist dies auch nicht nötig. Für allgemeine P2P-Anwendungen ist die Funktionalität durch die nicht unterstützte Replikation jedoch zu stark beschränkt. Für PACS wurde deshalb eine entsprechende Alternative entwickelt, die im Folgenden vorgestellt wird.

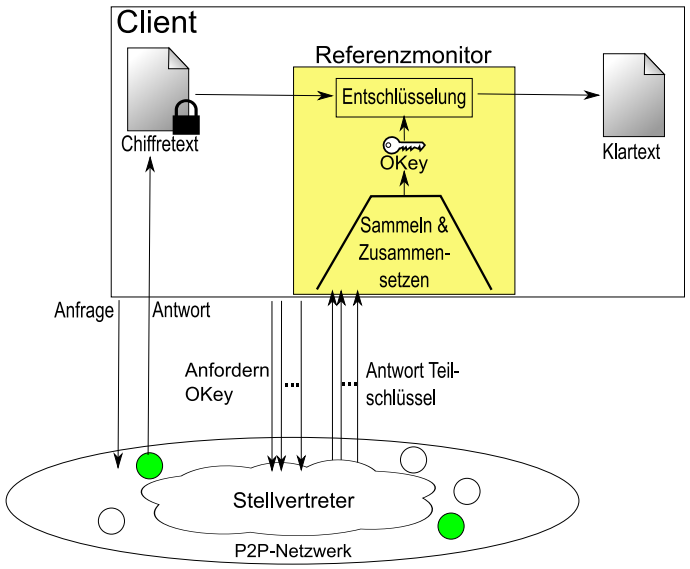


Abbildung 3.22: Kombination clientseitiger und gruppenbasierter Durchsetzung in PACS

3.8 Clientseitige Durchsetzung der Privilegien

Die clientseitige Durchsetzung erfüllt alle Anforderungen, die an eine verteilte Zugriffskontrolle für P2P-Datenbanken gestellt werden. Die grundlegende Idee des für PACS entwickelten Mechanismus ist eine Kombination aus clientseitiger und gruppenbasierter Durchsetzung, wie sie bereits vom Autor in [SHS09] vorgestellt wurde. Die Durchsetzung der Privilegien und somit das Entschlüsseln der Datenobjekte erfolgt hierbei auf dem Client. Die Privilegienverwaltung und damit die Schlüsselverteilung wird von einer Gruppe von Stellvertretern unter Verwendung eines Schwellensignaturverfahrens durchgeführt. Die Stellvertreter entschlüsseln also nicht die Daten, sondern lediglich den Schlüssel. Dies ist schematisch in Abbildung 3.22 dargestellt. Wie dort ersichtlich, erhält der Client vom P2P-Netzwerk nur ein verschlüsseltes Datenobjekt. Den Schlüssel zum Entschlüsseln des Datenobjekts $OKey$ muss er bei den Stellvertretern anfordern. Jeder der Stellvertreter besitzt hierbei nur einen Teilschlüssel. Durch Kombination der zugesandten Teilschlüssel kann der Client aus diesen den $OKey$ zusammensetzen und die Entschlüsselung des Datenobjekts vornehmen. Durch diese Kombination von clientseitiger und gruppenbasierter Durchsetzung ist es möglich, die größere Flexibilität gruppenbasierter Durchsetzungsverfahren zu erhalten und so administrative Delegation zu unterstützen. Gleichzeitig bleibt aber der hohe Aufwand der clientseitigen Durchsetzung beim Entzug von Privilegien bestehen (siehe Kapitel 2.5.2). In der von PACS gewählten Lösung übernimmt die Gruppe der Stellvertreter gemeinsam die Schlüsselerzeugung und Schlüsselverteilung. Die Verschlüsselung des Datenobjekts übernimmt im Normalfall der Client[3].

Die Kombination von gruppenbasierter und clientseitiger Durchsetzung vermindert die Flexibilität des Ansatzes gegenüber reiner gruppenbasierten Durchsetzung minimal. So ist es nicht möglich, Autorisierungen von Benutzern aufgrund von Kontextinformationen vorzunehmen. Da die Auswertung von Attributen während der XACML Policy Evaluation genau diese Kontextinformationen benutzt, können diese Attribute in XACML nicht unterstützt

[3]Ausnahme kann hierbei die erneute Verschlüsselung eines Datenobjekts aufgrund des Widerrufs eines Privilegs darstellen (siehe Kapitel 3.8.5.2).

werden. Jede Änderung der Kontextinformationen kann einen Entzug von Privilegien für bestimmte Benutzer zur Folge haben. Entsprechend müssen bei diesen Änderungen betroffene Datenobjekte mit einem neuen Schlüssel verschlüsselt werden. Nur so wird der Entzug der Privilegien wirksam. Die Durchsetzungskomponente hat keine Möglichkeit, diese Änderungen der Kontextinformationen zu überwachen und entsprechend zu reagieren, da sich diese Informationen außerhalb des Zuständigkeitsbereichs der Zugriffskontrolle befinden. Sollte ein Anwendungsgebiet die Attributevaluation zwingend erfordern, kann ein Wrapper diesen Attributwert auf eine Berechtigung in PACS abbilden. Auf diese Weise kann PACS Änderungen an diesem Attributwert nachvollziehen und entsprechend die Erzeugung neuer Schlüssel und die Neuverschlüsselung der Daten anstoßen.

Die in der clientseitigen Durchsetzung benutzten kryptographischen Methoden sind als sicher bewiesen. Die Kombination dieser Methoden in PACS nutzt keine weitergehenden Informationen und gibt auch keine weiteren Informationen preis. Deshalb ist auch die Kombination der kryptographischen Methoden sicher.

3.8.1 Speicherung der Datenobjekte

Die Schnittstelle zur Speicherung und Abfrage der durch die Zugriffskontrolle zu schützenden Daten stellt das P2P-Modul dar (siehe Kapitel 3.1.3). Da PACS zur Verschlüsselung und Entschlüsselung Zugriff auf diese Datenobjekte benötigt, muss diese Verbindung genauer erläutert werden. Das P2P-Modul stellt PACS die beiden Methoden $put(key, value)$ und $get(key)$ zum Speichern und Abfragen von Daten zur Verfügung. Wichtig ist hier die Unterscheidung zum Privilegienspeicher. Während dort der Schlüssel der Privilegien nicht eindeutig sein muss, ist hier die Eindeutigkeit des Schlüssels key bei den Datenobjekten zwingend erforderlich. Grund hierfür ist, dass PACS im Unterschied zu Privilegien bei Datenobjekten keinerlei Wissen über deren Inhalt oder Aufbau besitzt. Der Schlüssel eines Datenobjekts wird deshalb auch als DatenID bezeichnet. Die DatenID besteht aus dem eindeutigen Peernamen und dem eindeutigen Bezeichner des Datenobjekts. Die ID des Datenobjekts do von Peer p ist deshalb definiert als $ID_{do_p} = ID_p + ID_{do}$. Daten, die von PACS verschlüsselt werden, werden mit einer selbst verifizierenden DatenID ID^{SV} gespeichert. Diese wird erzeugt, indem ein Hashwert über den Inhalt des Datenobjekts und des Objektheaders $head_{do_p}$ erzeugt wird $ID^{SV}_{do_p} = h(head_{do_p} + do)$, wobei h eine kollisionsresistente Hashfunktion wie z.B. SHA-1 darstellt. Hierdurch wird gewährleistet, dass ein anfragender Peer anhand des Schlüssels ID^{SV} und des Datenobjekts prüfen kann, ob der Inhalt des Datenobjekts nach dem Speichern verändert wurde. Nur falls die erneute Anwendung der Hashfunktion die gleiche selbst verifizierende DatenID ID^{SV} ergibt, ist die Überprüfung erfolgreich. Deshalb werden diese Objektbezeichner auch selbst verifizierend genannt. Ein selbst verifizierender Objektbezeichner dokumentiert die Unversehrtheit des Datenobjekts, auf das er verweist. Die Peers des Datenspeichers stellen sicher, dass nur Datenobjekte gespeichert werden, die einen korrekten selbst verifizierenden ID^{SV} Schlüssel besitzen. Die Struktur der durch PACS geschützten Datenobjekte ist in Abbildung 3.23 dargestellt. Der Inhalt des Datenobjekts ist verschlüsselt, während der Header des Datenobjekts nicht verschlüsselt ist. Der Header enthält neben der DatenID ID_{do_p} den Bezeichner des DGIO (siehe Kapitel 3.8.3.2) und die Versionsnummer des Objektschlüssels $V_{OKey_{do_p}}$. Diese Information wird benötigt, um eine berechtigte Löschung von Datenobjekten nachvollziehen zu können. Der selbst verifizierende Schlüssel ID^{SV}, unter dem die Datenobjekte gespeichert sind, verändert sich bei jeder Datenobjektmodifikation. Deshalb bedarf es einer speziellen Löschoperation, um veraltete Datenobjekte zu löschen. Das Datenobjekt do_p kann durch den

Abbildung 3.23: Die Struktur des Datenobjekts do_p bei clientseitiger Durchsetzung

Aufruf der Methode $put(ID^{SV}_{do_p}, DGIO)$ gelöscht werden, wenn folgende Bedingungen gelten:

1. Beim Datenobjekt handelt es sich um das gültige DGIO, das auch im Header des zu löschenden Datenobjekts do_p vermerkt ist. D.h. ID_{DGIO} von $head_{do_p}$ ist gleich der ID_{DGIO} des DGIO.

2. Beim DGIO handelt es sich um ein neues oder aktualisiertes DGIO Objekt. Die dort für das Datenobjekt do_p enthaltene Versionsnummer ist größer als die im Header von do_p gespeicherte ($V_{OKey_{do_p}}$ von $head_{do_p} < V_{OKey_{do_p}}$ von $DGIO$).

Nur wenn beide Bedingungen erfüllt sind, wird der zuständige Peer das Datenobjekt do_p unter dem angegebenen Schlüssel $ID^{SV}_{do_p}$ löschen. Die Korrektheit des beim *put* Aufruf übergebenen DGIO wird zuvor durch Überprüfen seiner Signatur und Vergleich des Objekts mit dem aus dem Privilegienspeicher abgefragten DGIO überprüft. Der Schlüssel $ID^{SV}_{do_p}$ des *put* Aufrufs zum Löschen eines Datenobjekts hält einer Überprüfung nicht stand, da $ID^{SV}_{do_p} \neq h(DGIO)$. Für den Fall einer Löschoperation findet die Überprüfung des selbst verifizierenden Schlüssels deshalb nicht statt.

3.8.2 Verschlüsselung von Datenobjekten

Jedes Datenobjekt do wird mit einem individuellen Objektschlüssel $OKey_{do_p}$ (Länge 128 Bit) symmetrisch AES verschlüsselt. Peer p ist der Dateneigentümer des Datenobjekts. Jeder Objektschlüssel $OKey_{do_p}$ besitzt eine Versionsnummer $V_{OKey_{do_p}}$. Der Dateneigentümer Peer p stellt den privaten Masterschlüssel $MKey_p^{-1}$ und den öffentlichen Schlüssel $MKey_p$ des asymmetrischen RSA Verschlüsselungsverfahrens. Diese werden zum Erzeugen des $OKey_{do_p}$ benötigt, denn der $OKey_{do_p}$ ist eine digitale Signatur der Konkatenation von ID_{do_p} und $V_{OKey_{do_p}}$.

$$OKey_{do_p} = SIG_{MKey_p^{-1}}(ID_{do_p} + V_{OKey_{do_p}}) = E_{MKey_p^{-1}}(h(ID_{do_p} + V_{DOKey_{do_p}}))$$

3.8.3 Die Gruppe der Stellvertreter

Die Verteilung und Erzeugung der Objektschlüssel übernimmt eine vom Benutzer festgelegte Gruppe von Peers, die Stellvertreter. Für die Gruppe der Stellvertreter gilt, dass die Anzahl der darin enthaltenen bösartigen Peers dem Anteil bösartiger Peers im Netzwerk entspricht. Es ist Aufgabe des Dateneigentümers, die Größe n der Gruppe der Stellvertreter entsprechend groß zu wählen. PACS setzt weiterhin voraus, dass $t \geq \frac{n}{2}$, wobei t der Größe der Teilmenge entspricht, die für eine erfolgreiche Ausführung der Schwellenkryptographie nötig ist.

Für diese Gruppe wird ein (t, n) schwellenkryptographisches Verfahren etabliert. Hierzu wird der private Masterschlüssel $MKey_p^{-1}$ gemäß Shamir's (t, n) Secret Sharing Technique auf die n Gruppenmitglieder verteilt. Die erhaltenen Teilschlüssel $s_i (1 \leq i \leq n)$ werden mit

dem öffentlichen Schlüssel des entsprechenden Stellvertreters d_i verschlüsselt und an diesen übermittelt. Auf demselben Weg werden die zur Wiederherstellung verlorengegangener Teilschlüssel erzeugten Teilteilschlüssel $s_{i,j}(i,j \in \{1,\ldots,n\})$ an die Stellvertreter übermittelt. Dabei erhält der Stellvertreter d_i die Teilteilschlüssel $[s_{1,i},\ldots,s_{n,i}]$. Es obliegt dem Stellvertreter, diesen Teilschlüssel und die ebenfalls erzeugten Teilteilschlüssel sicher zu verwahren und sie Dritten nicht zugänglich zu machen.

Die Stellvertreter besitzen ein spezielles Privileg, um das korrekte erneute Verschlüsseln eines Datenobjekts bei einem Widerrufen von Privilegien zu überprüfen. Wurde ein Stellvertreter hierzu ausgewählt, erhält er kurzzeitig Zugriff auf das von ihm zu schützende unverschlüsselte Datenobjekt. Die korrekte Ausführung der Verschlüsselung und deren Überprüfung wird protokolliert und kann deshalb von autorisierten Benutzern überprüft werden. Trotzdem sollte die Auswahl der Stellvertreter entsprechend gewissenhaft vorgenommen werden. Es bietet sich an, Peers von Benutzern zu wählen, die schon Leseberechtigungen auf entsprechende Datenobjekte besitzen.

3.8.3.1 Schlüsselverwaltung

Jede Gruppe von Stellvertretern besitzt ihr eigenes Masterschlüsselpaar $MKey_p^{-1}$ und $MKey_p$, das vom Peer, der eine Gruppe erzeugt hat (in diesem Fall p), gestellt wird. Der private Masterschlüssel $MKey_p^{-1}$, wird wie schon beschrieben, in Form von Teilschlüsseln auf die Stellvertreter verteilt. Nach dieser Initialisierung ist diese Gruppe von Stellvertretern für die Erzeugung und Verteilung der Objektschlüssel $OKey$ verantwortlich. Die Objektschlüssel $OKey$ werden hierbei nicht von den Stellvertretern gespeichert, vielmehr werden sie für jede Anfrage neu erstellt.

Ein Objektschlüssel $OKey_{do_p}$ ist die Signatur der Verknüpfung von Datenobjektname mit der Versionsnummer des Objektschlüssels $SIG_{MKey_p^{-1}}(ID_{do_p} + V_{OKey_{do_p}})$. Die Versionsnummer und der Datenobjektname sind im DGIO gespeichert; so können die Stellvertreter durch ihre Teilschlüssel s_i Teilsignaturen $SIG_{s_i}((ID_{do_p} + V_{OKey_{do_p}})$ erzeugen. Durch die Sammlung von t Teilsignaturen verschiedener Stellvertreter kann daraus eine vollständige Signatur $SIG_{MKey_p^{-1}}(ID_{do_p} + V_{OKey_{do_p}})$ zusammengesetzt werden, die dem $OKey_{do_p}$ entspricht.

Zum Erhalt eines Objektschlüssels $OKey_{do_p}$ muss der Anfragende mindestens t Stellvertreter der Gruppe kontaktieren und den Schlüssel anfordern. Die Stellvertreter prüfen die Autorisierung des Benutzers und senden im Fall eines positiven Befundes eine Teilsignatur SIG_{s_i} zurück. Aus diesen Teilsignaturen setzt der Anfragende dann den Objektschlüssel $OKey_{do_p}$ zusammen. Dieser wird vom Anfragenden für weitere Anfragen zwischengespeichert, um die Netzwerklast zu reduzieren und die Stellvertreter zu entlasten.

Da die Objektschlüssel für jede Anfrage neu erstellt werden, geschieht die Erzeugung neuer Objektschlüsselversionen implizit, wenn die Versionsnummer des Schlüssels $V_{OKey_{do_p}}$ verändert wird.

3.8.3.2 Das Delegation Group Information Object (DGIO)

Alle Informationen zu einer Gruppe von Stellvertretern werden in einem DGIO gespeichert. Die DGIOs werden, aufgrund ihrer hohen Bedeutung, im Privilegienspeicher abgelegt. In PACS kann jeder Peer mehrere Gruppen von Stellvertretern besitzen. Er muss nur jeder dieser Gruppen ein eigenes Masterschlüsselpaar $[MKey_p^{-1}, MKey_p]$ bereitstellen. Zudem muss er dafür sorgen (z.B. als Anhang an seine Export Policy), dass jedem Benutzer des Netzwerks

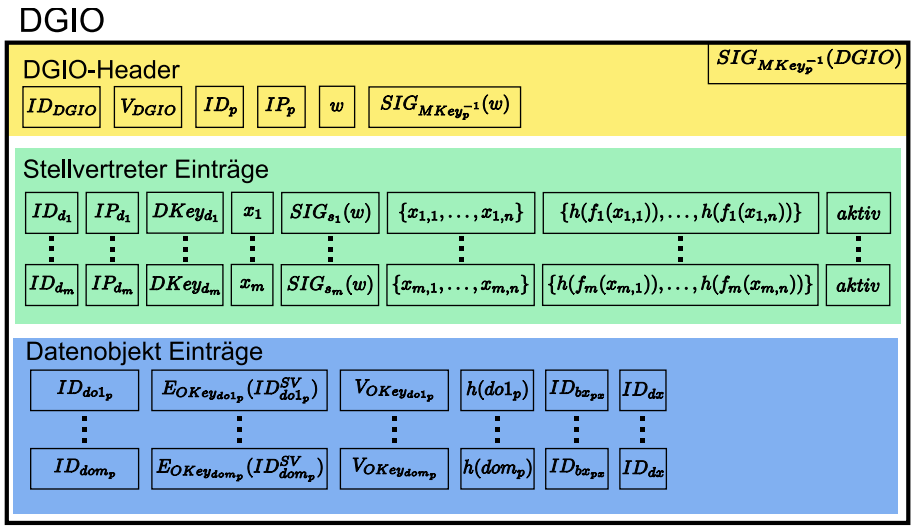


Abbildung 3.24: Die Struktur des DGIO für Peer p

bekannt ist, welche der Stellvertretergruppen des Peers für welches Datenobjekt zuständig sind. Zur besseren Lesbarkeit wird im Folgenden davon ausgegangen, dass jeder Peer exakt eine Gruppe von Stellvertretern besitzt. Zudem besteht ein allgemein bekanntes Namensschema für das DGIO, sodass jeder Peer den Schlüssel ID_{DGIO} des DGIO Objekts ermitteln kann. ID_{DGIO} kann z.B. dem Peernamen entsprechen. Zudem muss der öffentliche Masterschlüssel $MKey_p$ an einem sicheren Ort hinterlegt werden. In PACS wird dieser in die Export Policy des Peers aufgenommen. Da diese vom Peer signiert ist und über eine Versionsnummer zur Verhinderung von Replay-Angriffen verfügt, ist dieser dort sicher verwahrt.

Das DGIO besteht, wie aus Abbildung 3.24 ersichtlich, aus einem Header und einem Datenteil. Teil des Headers ist die Signatur $SIG_{MKey_p^{-1}}(DGIO)$ des gesamten DGIO (exklusiv der Signatur selbst). Diese verhindert unautorisierte Änderungen am DGIO. ID_P ist der eindeutige Peername von Peer p, zu dem dieses DGIO gehört und IP_P ist seine zuletzt aktuelle IP Adresse. w ist die Musternachricht und die ihr entsprechende Signatur $SIG_{MKey_p^{-1}}(w)$, die für die Verifizierung von Teilsignaturen der Gruppenmitglieder benötigt wird. V_{DGIO} schließlich ist die Versionsnummer des DGIO. Diese wird zur Erkennung veralteter DGIO Objekte benötigt.

Der Datenteil untergliedert sich weiter in einen Teil für Stellvertreter Einträge und einen Teil für Datenobjekt Einträge. Für jeden Stellvertreter gibt es einen Eintrag, bestehend aus dem eindeutigen Peernamen ID_{d_i}, seiner IP-Adresse IP_{d_i}, unter der er erreichbar ist, sowie dem öffentlichen Schlüssel des Peers $DKey_{d_i}$. Dadurch kann mit allen Stellvertretern direkt kommuniziert werden. Zudem wird die x Koordinate x_i des Teilschlüssels, die diesem Peer zugewiesen wurde, sowie die x Koordinaten $\{x_{i,1},\ldots,x_{i,n}\}$ der Teilteilschlüssel dieses Teilschlüssels gespeichert. Die Teilsignatur mit der Musternachricht $SIG_{s_i}(w)$ dient der Verifikation der Teilsignaturen (siehe Kapitel 2.3.3.3). Zusätzlich werden die Hashwerte $\{h(f_i(x_{i,1})),\ldots,h(f_i(x_{i,n}))\}$ der Teilteilschlüssel aufgeführt, die der Verifikation der Teilteilschlüssel dienen.

Für jedes Datenobjekt, für das die Gruppe der Stellvertreter für die Durchsetzung der Privilegien verantwortlich ist, besitzt das DGIO einen eigenen Eintrag. Er enthält den eindeu-

tigen Namen des Datenobjekts ID_{do_p} sowie die aktuelle Versionsnummer des Objektschlüssels $V_{OKey_{do_p}}$ für dieses Datenobjekt. Zudem enthält der Eintrag den aktuellen selbst verifizierenden Schlüssel des Objekts $ID^{SV}_{do_p}$, unter dem das Objekt im Datenspeicher abgerufen werden kann. Durch dessen Speicherung im DGIO ist sichergestellt, dass nur autorisierte Benutzer neue Versionen des Datenobjekts erstellen und im DGIO registrieren können. Zusätzlich ist der $ID^{SV}_{do_p}$ Objektschlüssel des Datenobjekts mit dem $OKey_{do_p}$ verschlüsselt, sodass nur autorisierte Benutzer diesen sehen können. Hinzu kommt der Hashwert des unverschlüsselten Datenobjekts $h(do_p)$. Mit dessen Hilfe können Veränderungen am Inhalt des Datenobjekts von jenen unterschieden werden, die durch eine erneute Verschlüsselung entstehen. Um Veränderungen am Datenobjekt nachprüfbar zu machen, wird die ID_{b_p} des Benutzers registriert, der zuletzt eine Änderung am Datenobjekt vorgenommen hat. Schließlich wird die ID_d des Stellvertreters aufgenommen, der die letzte für dieses Datenobjekt vorgenommene erneute Verschlüsselung überprüft hat. Dies ist für die Feststellung hierbei vorgenommener Manipulationen erforderlich.

Schutz von DGIOs Aufgrund der sensiblen Informationen in DGIOs werden diese im Privilegienspeicher abgelegt. Ähnlich wie bei Export Policies achten die Knoten des Privilegienspeichers darauf, dass gespeicherte DGIOs auch inhaltlich korrekt sind und nicht von unautorisierten Peers manipuliert wurden.

Gutartige Peers speichern deshalb nur DGIOs, die eine gültige Signatur des zu diesem DGIO gehörenden $MKey_p^{-1}$ besitzen. Hierzu beziehen sie den entsprechenden öffentlichen Schlüssel $MKey$ aus der Export Policy des Peers, zu dem ein DGIO gehört. Ist die Signatur nicht korrekt oder gehört das DGIO Objekt nicht zum vorgegebenen Peer, wird die Speicherung verweigert. Ist die Signatur korrekt, wird weiter überprüft, ob schon ein DGIO mit gleichem Bezeichner ID_{DGIO} vorhanden ist. Ist dies der Fall, wird die Speicherung des DGIO nur vorgenommen, wenn die Versionsnummer V_{DGIO} des zu speichernden DGIO größer ist, als die des schon vorhandenen DGIO. Durch diese Überprüfungen wird sichergestellt, dass erstens nur gültige DGIOs aufgenommen werden und zweitens, dass alte DGIOs nicht neuere DGIOs überschreiben können.

3.8.3.3 Gruppenänderungen

Stellvertreter können wie alle anderen Peers jederzeit aus dem Netzwerk ausscheiden, ohne zuvor die anderen Gruppenmitglieder darüber zu informieren. Wie schon bei den Replikationsgruppen in unstrukturierten Netzwerken beschrieben (siehe Kapitel 3.5.6.4), müssen die Gruppenmitglieder deshalb periodisch kontrollieren, ob die anderen Gruppenmitglieder noch erreichbar sind. Hier werden nun die kryptographischen Details der Gruppenänderung beschrieben. Entdeckt ein Peer d_i der Gruppe die Abwesenheit eines anderen Gruppenmitglieds d_j, kontaktiert er die verbliebenen Gruppenmitglieder, um von ihnen die Zustimmung für folgende Vorgänge zu erhalten:

1. Stellvertreter d_j ist nicht länger im Netzwerk und sollte deshalb als gelöscht markiert werden.

2. Peer d_{n+1} ist der neue Stellvertreter, der d_j ersetzt.

Für die Zustimmung zu diesen Aktionen ist das Quorum (wiederum t aus n) der verbleibenden Teilnehmer erforderlich. Die Zustimmung eines Peers erfolgt, indem er das neu modifizierte

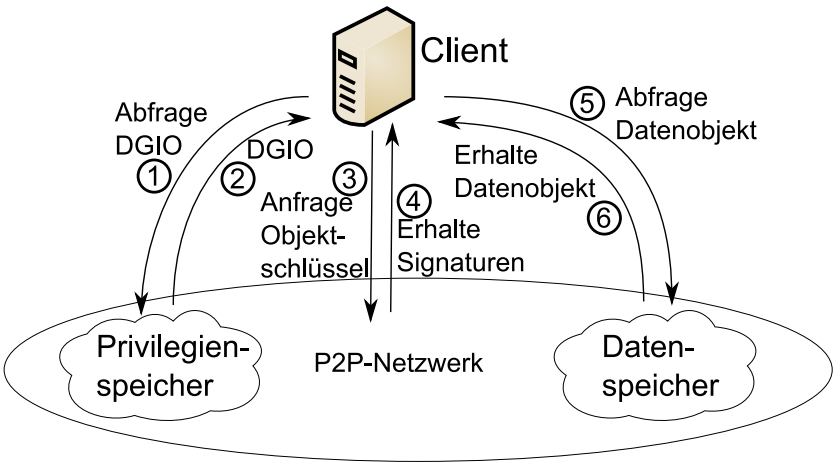


Abbildung 3.25: Ablauf einer Datenanfrage bei clientseitiger Durchsetzung in PACS

DGIO mit seinem Teilschlüssel signiert und diese Teilsignatur an d_i sendet. Erhält d_i so t gültige Teilsignaturen, kann er daraus die vollständige Signatur $SIG_{MKey_p^{-1}}(DGIO)$ zusammensetzen. Diese Signatur wird nun im DGIO Objekt gespeichert und das neu signierte DGIO an die Stellvertreter übermittelt. Nun startet der Prozess, um den Teilschlüssel s_j, der durch Verlust von Peer d_j verloren ging, für den neuen Peer d_{n+1} wiederherzustellen (siehe Kapitel 2.3.3.1). Nach der Wiederherstellung von s_j wird das DGIO im Privilegienspeicher gespeichert.

Wie bereits erwähnt, wird d_j lediglich als gelöscht markiert und nicht aus dem DGIO gelöscht. Damit wird er als zukünftiger Ersatz für ausfallende Stellvertreter ausgeschlossen. So wird verhindert, dass ein Peer durch häufiges Verlassen und Beitreten zum Netzwerk in den Besitz weiterer Teilschlüssel kommen kann. In der von PACS gewählten Lösung bleiben bei einem Gruppenwechsel die Teilschlüssel und Teilteilschlüssel bestehen. D.h. der Teilschlüssel s_j, den d_j besitzt, ist nach wie vor gültig. Tritt Peer d_j dem Netzwerk zu einem späteren Zeitpunkt wieder bei, benötigt er entsprechend nur $t-1$ Teilsignaturen SIG_{s_i} mit $i \neq j$, um eine vollständige Signatur $SIG_{MKey_p^{-1}}$ zusammenzusetzen zu können. Häufige Gruppenwechsel schwächen also die Sicherheit der Schwellenkryptographie und Secret Sharing Technique. Proaktives Secret Sharing [HJKY95] verhindert dieses, indem die Teilschlüssel und Teilteilschlüssel häufig ausgetauscht werden. Für die Anforderungen von PACS ist dies aber übertrieben. Hier reicht es aus, wenn der Erzeuger einer Stellvertretergruppe gemäß seinen eigenen Sicherheitsanforderungen den Secret Sharing Algorithmus von Zeit zu Zeit reinitialisiert und somit neue Teil- und Teilteilschlüssel für die Stellvertretergruppe generiert. Die Erzeugung neuer Teil- und Teilteilschlüssel verändert hierbei nur das DGIO und die Stellvertreter. Die verschlüsselten Daten hingegen müssen nicht neu verschlüsselt werden, da sich der Masterschlüssel $MKey^{-1}$ und somit auch die Objektschlüssel $OKey$ nicht ändern.

3.8.4 Ablauf einer Datenanfrage in PACS

Der Ablauf einer Datenanfrage bei clientseitiger Durchsetzung unterscheidet sich grundlegend von jener der serverseitigen Durchsetzung. Der Ablauf einer Datenanfrage ist in Abbildung 3.25 dargestellt und läuft wie folgt ab:

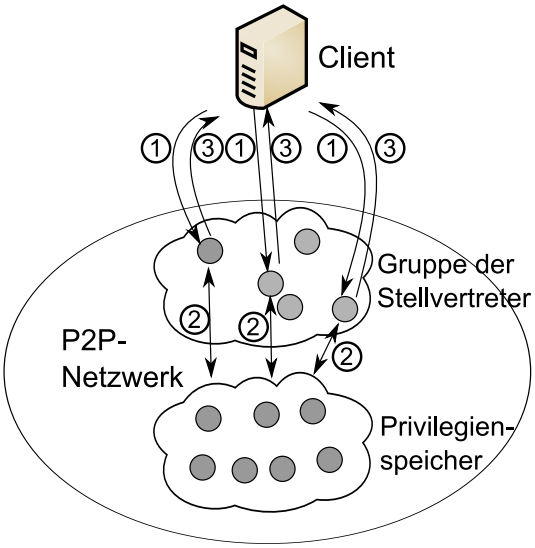


Abbildung 3.26: Die Gruppe der Stellvertreter bearbeitet eine Anfrage

1. Der Client sendet eine Anfrage nach dem DGIO an den Privilegienspeicher.
2. Der Client erhält das DGIO zurück.
3. Der Client sendet nun Anfragen an die Stellvertreter, ihm den Objektschlüssel für das Datenobjekt zu liefern.
4. Der Client erhält Teilsignaturen SIG_{s_i} der Stellvertreter zurück. Er setzt diese Teilsignaturen zur vollständigen Signatur $SIG_{MKey^{-1}}$ zusammen, die dem Objektschlüssel entspricht.
5. Nun entschlüsselt er den im DGIO vermerkten, selbst verifizierenden Schlüssel ID^{SV} des Datenobjekts und sendet eine Anfrage nach genau diesem Schlüssel an den Datenspeicher.
6. Der Client erhält das Datenobjekt zurück und entschlüsselt es nun mit dem Objektschlüssel. Damit ist die Anfrage abgeschlossen.

Eine detailliertere Darstellung der Schritte drei und vier wird in Abbildung 3.26 gezeigt. Der Anfragende muss mindestens t Stellvertreter nach dem Objektschlüssel fragen (Schritt 1). Jeder Stellvertreter prüft für sich, ob der Anfragende über die hierzu erforderlichen Privilegien verfügt. Für diese Prüfung greifen die Stellvertreter auf die Privilegien des Privilegienspeichers zurück (Schritt 2). Die Überprüfung verläuft hierbei deckungsgleich mit dem schon in Kapitel 3.6 dargestellten Vorgehen. Nur wenn diese Prüfung erfolgreich war, wird der Stellvertreter dem Anfragenden seine Teilsignatur SIG_{s_i} zurücksenden (Schritt 3). Besitzt der Anfragende den Objektschlüssel schon, weil er sich schon von einer früheren Anfrage im Zwischenspeicher befindet, entfallen die Schritte 3 und 4 in Abbildung 3.25.

3.8.5 Durchsetzung der Privilegien durch Datenverschlüsselung

Die Durchsetzung der Privilegien geschieht durch Verschlüsselung der Daten (Kapitel 3.8.2) und Verteilung der Schlüssel an berechtigte Peers (Kapitel 3.8.3.1). Im Folgenden wird erläu-

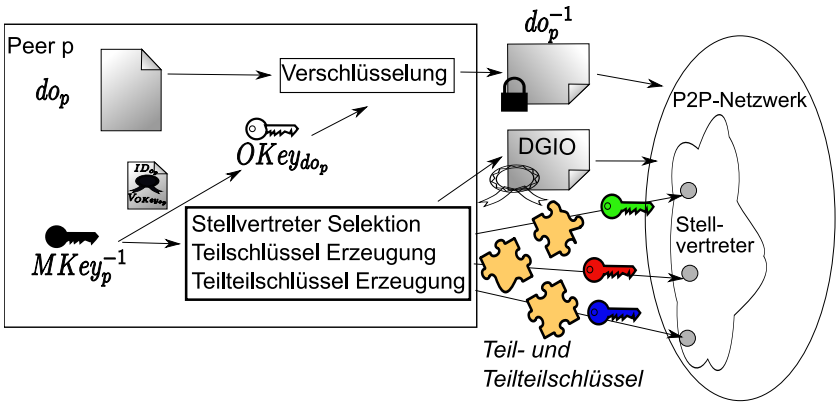


Abbildung 3.27: Vereinfachte Darstellung der initialen Verschlüsselung eines Datenobjekts

tert, wann eine solche Verschlüsselung der Daten angestoßen werden muss, um die Durchsetzung der Privilegien zu garantieren. Für die Verschlüsselung der Daten zur Durchsetzung der Privilegien gilt es drei Fälle zu unterscheiden:

1. Die initiale Verschlüsselung, wenn ein neues Datenobjekt ins Netzwerk eingefügt wird, das durch Zugriffskontrolle geschützt werden soll.
2. Die erneute Verschlüsselung der Daten durch das Widerrufen eines Privilegs.
3. Verändern eines gespeicherten verschlüsselten Datenobjekts.

3.8.5.1 Initiale Verschlüsselung

Die initiale Verschlüsselung ist in vereinfachter Form in Abbildung 3.27 dargestellt. Ein Peer p möchte ein neues Datenobjekt do_p in das Netzwerk einfügen und es entsprechend durch Zugriffskontrolle schützen lassen. Hierzu muss Peer p zunächst eine Gruppe von Stellvertretern erzeugen, falls er noch keine solche besitzt oder die existierende für dieses Datenobjekt nicht geeignet ist. p erzeugt die Masterschlüsselpaare $[MKey_p, MKey_p^{-1}]$. Mit $MKey_p^{-1}$ kann p nun den Objektschlüssel $OKey_{do_p}$ für das Datenobjekt do_p erzeugen. Hierzu signiert p den Bezeichner des Datenobjekts ID_{do_p} zusammen mit der Versionsnummer $V_{OKey_{do_p}}$. Da dies der Start der Verschlüsselung ist, gilt $V_{OKey_{do_p}} = 0$. Der initiale Objektschlüssel ist deshalb $OKey_{do_p} = SIG_{MKey_p^{-1}}(ID_{do_p} + 0)$. Mit diesem so generierten Objektschlüssel wird das Datenobjekt verschlüsselt. Das Resultat ist das Chiffredatenobjekt $do_p^{-1} = E_{OKey_{do_p}}(do_p)$. Dem Chiffredatenobjekt wird nun noch der Header (Kapitel 3.8.1) angefügt. p ermittelt anschließend den selbst verifizierenden Schlüssel $ID_{do_p}^{SV} = h(head_{do_p} + do_p^{-1})$ für das Chiffredatenobjekt und speichert dieses durch Aufruf von $put(ID_{do_p}^{SV}, (head_{do_p} + do_p^{-1}))$ im Datenspeicher.

Falls p eine neue Stellvertretergruppe erzeugt hat, muss das DGIO noch erzeugt werden. Nach der Auswahl der n Stellvertreter werden wie zuvor beschrieben aus dem privaten Masterschlüssel die Teil- und Teilteilschlüssel erzeugt und an die Stellvertreter verteilt. Anschließend erfolgt die Erzeugung des DGIO, das dann im Privilegienspeicher gespeichert wird. Der $MKey$ wird in die Export Policy des Peers aufgenommen, welche ebenfalls im Privilegienspeicher abgelegt wird.

Ist die Stellvertretergruppe bereits vorhanden, muss die Information für das neue Datenobjekt do_p in das bestehende DGIO eingefügt werden. Das so erweiterte DGIO wird anschlie-

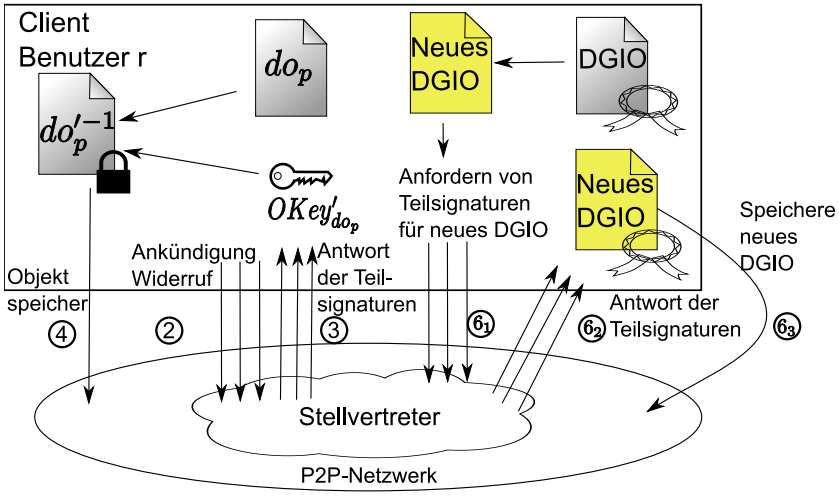


Abbildung 3.28: Vereinfachte Darstellung der erneuten Verschlüsselung eines Datenobjekts

ßend im Privilegienspeicher gespeichert. Autorisierte Benutzer haben nun die Möglichkeit, auf das gespeicherte neue Datenobjekt do_p nach dem in Kapitel 3.8.4 geschilderten Vorgehen zuzugreifen.

3.8.5.2 Widerrufen eines Privilegs

Die erneute Verschlüsselung eines Datenobjekts do_p ist nötig, falls ein Privileg für dieses Datenobjekt durch einen Benutzer r entzogen wurde. Ein nicht lokaler Benutzer, der die Berechtigung besitzt, Privilegien an andere Benutzer weiterzureichen, hat automatisch Leseberechtigungen an diesem Objekt. Dies ist eine Bedingung clientseitiger Durchsetzung von Privilegien. Hierdurch ist gewährleistet, dass der Benutzer, der ein Privileg entzieht, den Objektschlüssel $OKey_{do_p}$ für das betreffende Objekt erhält. Nur so kann der Benutzer das Datenobjekt do_p im Datenspeicher abfragen und erneut verschlüsseln. Bei einer erneuten Verschlüsselung des Datenobjekts handelt es sich faktisch um die Aktualisierung von do_p und damit um eine Schreiboperation, allerdings mit gleichzeitigem Objektschlüsselwechsel. Besitzt Benutzer r auch eine Schreibberechtigung, kann er die erneute Verschlüsselung des Datenobjekts selbst vornehmen. Damit lässt sich der Vorgang abkürzen. Dies entspricht dann dem Vorgehen bei einer Änderung des Datenobjekts. Allerdings wird zum Verschlüsseln des Datenobjekts zuvor ein neuer Schlüssel bei den Stellvertretern angefordert.

Ist der Dateneigentümer im Netzwerk verfügbar, wird die erneute Verschlüsselung der Datenobjekte bevorzugt durch ihn vorgenommen, denn er besitzt schon alle hierzu nötigen kryptographischen Schlüssel. Besitzt weder r eine Schreibberechtigung für do_p, noch ist Peer p im Netzwerk erreichbar, muss r das erneut verschlüsselte Objekt von einem Stellvertreter verifizieren lassen. Hierzu sind die Stellvertreter (Kapitel 3.8.3) im Falle eines Widerrufs eines Privilegs berechtigt. Der genaue Ablauf dieser erneuten Verschlüsselung ist nachfolgend in den Punkten eins bis sechs beschrieben. Teile dieses Vorgangs sind zum besseren Verständnis zusätzlich in Abbildung 3.28 dargestellt.

1. Vor dem Widerrufen eines Privilegs muss der Benutzer r, der das Privileg widerruft, das DGIO des Datenobjekts anfordern. Anschließend wird der Objektschlüssel $OKey_{do_p}$ von

den Stellvertretern angefordert, sofern dieser noch nicht lokal zwischengespeichert ist. Hat der Benutzer den Objektschlüssel $OKey_{do_p}$ erhalten, wird er das Datenobjekt do_p^{-1} anfordern und sobald erhalten entschlüsseln, um das Klartextobjekt do_p zu erhalten.

2. Benutzer r bereitet nun den Widerruf des zu widerrufenden Privilegs vor, indem er diesen bei den Stellvertretern ankündigt. Diese können die Rechtmäßigkeit des Widerrufs feststellen, indem sie folgende Sachverhalte prüfen:

   (a) Handelt es sich beim zu widerrufenen Privileg um ein gültiges Privileg?

   (b) Ist r für diesen Widerruf autorisiert?

   Anschließend nimmt Benutzer r den Widerruf, wie in Kapitel 3.4.2 und 3.4.4 beschrieben, vor. Punkt (b) kann deshalb auch überprüft werden, indem auf den Widerruf des Privilegs im Privilegienspeicher gewartet wird.

3. Stellvertreter, die diesen Widerruf akzeptieren, senden an r einen neuen Objektschlüssel $OKey'_{do_p}$ für die erneute Verschlüsselung des Datenobjekts do_p. Diese erneute Verschlüsselung ist nötig, da der Benutzer, dem das Privileg entzogen wurde, den Objektschlüssel $OKey_{do_p}$ eventuell kennt und zwischengespeichert hat. Damit könnte er nach wie vor das Datenobjekt do_p^{-1} entschlüsseln und so das Datenobjekt do_p im Klartext erhalten.

   Die neue Version des Objektschlüssels $OKey'_{do_p}$ wird durch die Stellvertreter erzeugt, indem sie die Versionsnummer $V_{OKey_{do_p}}$ inkrementieren. Die Teilsignatur $SIG_{s_i}(ID_{do_p} + V_{OKey_{do_p}} + 1)$ wird anschließend verschlüsselt an r gesendet.

4. Sobald r, t gültige Teilsignaturen empfangen hat, kann er daraus den neuen Objektschlüssel $OKey'_{do_p}$ zusammensetzen, da $OKey'_{do_p} = SIG_{MKey_p^{-1}}(ID_{do_p} + V_{OKey_{do_p}} + 1)$. r verschlüsselt das Datenobjekt mit dem neuen Objektschlüssel, $do_p'^{-1} = E_{OKey'_{do_p}}(do_p)$. Dieses neue verschlüsselte Objekt wird mit seinem neuen selbst verifizierenden Schlüssel im Datenspeicher durch den Aufruf von $put(ID^{SV}_{do'_p}, (head'_{do_p}, do_p'^{-1})$ gespeichert.

5. Nach der Speicherung informiert r die Stellvertreter über die Aktualisierung, indem er ihnen $ID^{SV}_{do'_p}$ zusendet. Die Stellvertreter überprüfen nun das erneut verschlüsselte Datenobjekt $do_p'^{-1}$. Hierzu bestimmen Sie einen Stellvertreter pr, der dies für sie verifiziert. Bevorzugt ist dieser ein Stellvertreter, der für das betroffene Datenobjekt über eine Leseberechtigung verfügt. Ansonsten wird ein Stellvertreter zufällig bestimmt, z.B. der Stellvertreter dessen ID der $ID^{SV}_{do'_p}$ am nächsten ist. An diesen Stellvertreter pr werden die Teilsignaturen $SIG_{s_i}(ID_{do_p} + V_{OKey_{do_p}} + 1)$ gesandt. Hierdurch ist er in der Lage, $OKey'_{do_p}$ zusammenzusetzen. Mit diesem entschlüsselt er $do_p'^{-1}$ und vergleicht dessen Hashwert $h(do'_p)$ mit dem Hashwert $h(do_p)$, der für das Datenobjekt do_p im DGIO gespeichert wurde. Sind die beiden Hashwerte identisch, ist die Verifikation der Verschlüsselung erfolgreich. Das Ergebnis dieser Überprüfung wird an alle Stellvertreter der Gruppe und r übermittelt.

6. Im Falle einer bestandenen Überprüfung wird der Eintrag für das Datenobjekt im DGIO aktualisiert. Diese Aktualisierung kann entweder von einem der Stellvertreter oder von r vorgenommen werden. Im Datenobjekteintrag für do_p wird hierbei Benutzer r und der

prüfende Stellvertreter *pr* vermerkt. Zudem wird die Versionsnummer des Objektschlüssels inkrementiert und der verschlüsselte selbst verifizierende Schlüssel aktualisiert. Da sich der Inhalt von do_p nicht verändert hat, bleibt der Hashwert $h(do_p)$ gleich.

Das aktualisierte DGIO muss vom privaten Masterschlüssel $MKey_p^{-1}$ signiert werden. Hierbei kommt wiederum das Schwellensignaturverfahren zum Einsatz. Senden t Stellvertreter ihre gültige Teilsignatur für das aktualisierte DGIO, kann daraus die erforderliche Signatur zusammengesetzt und das mit dieser Signatur versehene neue DGIO im Privilegienspeicher abgelegt werden.

War die Überprüfung der Verschlüsselung (Punkt 5) nicht erfolgreich, kann dies daran liegen, dass ein bösartiger Peer zur Verifizierung ausgewählt wurde. Ist r von der Korrektheit seiner Verschlüsselung überzeugt, kann er deshalb eine erneute Prüfung verlangen. Die Ergebnisse dieser Prüfungen werden kumuliert; die Mehrheit der Ergebnisse bestimmt das Gesamtergebnis. Zur Aufhebung des ersten Ergebnisses müssen also zwei gegensätzliche Ergebnisse vorhanden sein.

Falls Benutzer r tatsächlich unberechtigte Manipulationen am Datenobjekt do_p vorgenommen hat und ein ebenfalls bösartiger Stellvertreter *po* dies bestätigt hat, kann diese unberechtigte Manipulation zweifelsfrei aufgedeckt werden, da der im DGIO gespeicherte Hashwert des Inhalts des unverschlüsselten Datenobjekts nicht mehr $h(do_p)$ entspricht. Der Benutzer und der Stellvertreter, die für dieses Datenobjekt vermerkt sind, werden daraufhin als bösartig überführt und können aus dem Netzwerk ausgeschlossen werden. Aufgrund der leichten Feststellbarkeit solcher Manipulationen ist deren Auftreten eher unwahrscheinlich.

Nach der Speicherung des neu verschlüsselten Datenobjekts $do_p'^{-1}$ und des aktualisierten DGIO kann das alte Datenobjekt do_p^{-1} von r aus dem Datenspeicher gelöscht werden. Hierzu ruft r die Methode $put(ID_{do_p}^{SV}, DGIO)$ auf, wobei $ID_{do_p}^{SV}$ der selbst verifizierende Schlüssel des zu löschenden Datenobjekts do_p^{-1} ist und DGIO das neue aktualisierte DGIO. Dies führt zum Löschen des Datenobjekts do_p unter dem Schlüssel $ID_{do_p}^{SV}$ (siehe Kapitel 3.8.1).

3.8.5.3 Veränderung eines Datenobjekts

Der Ablauf bei einer Veränderung eines Datenobjekts ist wie folgt (siehe Abbildung 3.29):

1. Wenn Peer p ein modifiziertes Datenobjekt do_p' speichern möchte, verschlüsselt er es mit demselben Objektschlüssel $OKey_{do_p}$, mit dem er das zuvor unverschlüsselte Datenobjekt do_p entschlüsselt hat. Anschließend erzeugt p den neuen, selbst verifizierenden Schlüssel $ID_{do_p'}^{SV}$, mit dem er das modifizierte verschlüsselte Objekt do_p' im Datenspeicher ablegt.

2. Nun informiert er die Stellvertreter über die Veränderung am Datenobjekt ID_{do_p}. Hierzu sendet er den Stellvertretern das DGIO mit aktualisiertem Eintrag für ID_{do_p} zu (insbesondere wird hierbei der Hashwert über den Inhalt des Datenobjekts auf $h(do_p')$ aktualisiert).

3. Die Stellvertreter prüfen, ob der Benutzer p die Berechtigung zur Modifikation des Datenobjekts do_p besitzt. Im Falle einer erfolgreichen Prüfung senden sie ihre Teilsignatur $SIG_{s_i}(DGIO)$ des aktualisierten DGIO an p zurück.

4. Sobald p t gültige Teilsignaturen erhalten hat, kann er daraus die vollständige Signatur des aktualisierten DGIO zusammensetzen und diese dem DGIO anfügen. Das nun signierte aktualisierte DGIO wird im Privilegienspeicher abgelegt.

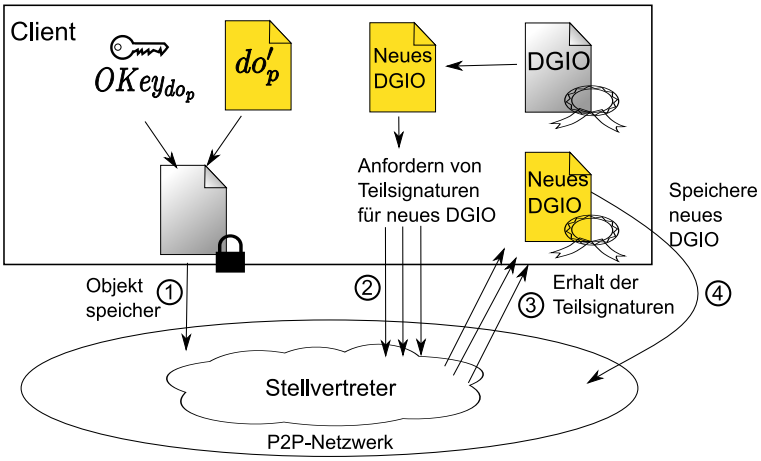


Abbildung 3.29: Veränderung eines Datenobjekts

3.8.6 Organisation des Privilegienspeichers

In PACS wird eine vollständige Replikation realisiert, d.h. die Daten sollen verfügbar bleiben, falls der Dateneigentümer das Netzwerk verlässt. Hieraus ergibt sich, dass auch die Speicherung von Berechtigungen und deren Durchsetzung auf andere Peers repliziert wird. Nur so bleiben im Falle eines Ausfalls des Dateneigentümers auch Daten, die durch eine Zugriffskontrolle geschützt werden, für andere Benutzer abrufbar.

Ein allgemeiner Privilegienspeicher ermöglicht eine von Grantor und Datenobjekt unabhängige Speicherung von Berechtigungen. Er garantiert die Auffindbarkeit und Gültigkeit von Privilegien und Benutzerzuordnungen, auch wenn der Grantor und der Datenobjekteigentümer sich nicht im Netzwerk befinden.

Dies ist Grundlage für die clientseitige Durchsetzung. Wie bei der serverseitigen Durchsetzung schon erläutert, müssen auch hier Export Policies und DGIOs im Privilegienspeicher abgelegt werden. Im Gegensatz zur serverseitigen Durchsetzung, ist es bei der clientseitigen Durchsetzung, die Replikation unterstützt, zudem nötig, alle Privilegien und Benutzerzuordnungen der globalen Ebene im Privilegienspeicher abzulegen.

Grundsätzlich ist es wünschenswert, Privilegien und Benutzerzuordnungen dort zu speichern, wo sie gebraucht werden. Dies ist bei clientseitiger Durchsetzung bei den Stellvertretern des Datenobjekteigentümers der Fall. Entsprechend müssen Privilegien immer in der Stellvertretergruppe des zugehörigen Datenobjekteigentümers gespeichert werden. Die Speicherung aller Privilegien an einem Datenobjekt auf denselben Peers bringt weitere Vorteile. So können diese Peers die Überprüfung, ob ein neues zu speicherndes Privileg korrekt ist, leichter vornehmen, da ihnen alle Privilegien schon lokal vorliegen. Die Stellvertreter müssen dadurch weder bei der Speicherung noch bei der Durchsetzung im Netzwerk nach Privilegien suchen. Dieser Vorteil wiegt die eventuell ungleiche Verteilung der zu speichernden Privilegien über das Netzwerk mehr als auf. Da durch die clientseitige Durchsetzung die Stellvertretergruppen schon vorhanden sind, wird die Speicherung der Privilegien bei beiden Lösungen an diese übertragen.

Für die Organisation des Privilegienspeichers kommen für die verwendeten Techniken DHT und unstrukturiertes Netzwerk verschiedene Möglichkeiten in Frage, die nun erläutert werden.

3.8.6.1 Speicherung und Abfrage von Export Policies und DGIOs

Bei der Speicherung von Export Policies gilt dasselbe Vorgehen wie bei der serverseitigen Durchsetzung (siehe Kapitel 3.7.2). Entsprechendes gilt auch für deren Abfrage.

Auch die DGIOs werden wie bei der serverseitigen Durchsetzung abgelegt. Da die clientseitige Durchsetzung die DGIOs benötigt, werden diese auch bei Verwendung eines DHT-Privilegienspeicher abgespeichert. Ihre Speicherung erfolgt hierbei unter ihrem eindeutigen Bezeichner ID_{DGIO}.

Für die Abfrage der DGIOs gelten die Ausführungen der serverseitigen Durchsetzung. Auch beim DHT-Privilegienspeicher reicht hierbei ein Stichprobentest mit fünf Stichproben aus, um die Korrektheit eines DGIO bei Anfragen zu garantieren.

3.8.6.2 Speicherung von Berechtigungen im unstrukturierten P2P-Privilegienspeicher

Bei der Ablage von Berechtigungen im unstrukturierten Privilegienspeicher wurde folgendes Vorgehen für Privilegien und Benutzerzuordnungen gewählt.

Privilegien Bei der Speicherung von Privilegien in einem unstrukturierten Netzwerk ist die Zuweisung der zu speichernden Informationen zu Replikationsgruppen entscheidend. Diese Replikationsgruppen entsprechen den Stellvertretergruppen der clientseitigen Durchsetzung.

Da Privilegien immer in der Replikations- bzw. Stellvertretergruppe des Datenobjekteigentümers gespeichert werden sollen, müssen zunächst die Mitglieder dieser Stellvertretergruppe ermittelt werden. Dies ist durch die Abfrage des gespeicherten DGIO des Datenobjekteigentümers vom Privilegienspeicher leicht möglich.

Wenn der Fall ausgeklammert wird, dass der Dateneigentümer sich im Netzwerk befindet, ist der Ablauf einer Privilegienspeicherung wie folgt. Das zu speichernde Privileg wird auf alle Mitglieder der Stellvertretergruppe des Datenobjekteigentümers abgelegt. Sofern der Grantor berechtigt ist, dieses Privileg zu erteilen, werden die gutartigen Peers der Stellvertretergruppe dieses Privileg abspeichern.

Benutzerzuordnungen und administrative Privilegien von Rollen Benutzerzuordnungen und administrative Privilegien von Rollen werden auf die gleiche Art und Weise im Privilegienspeicher abgelegt wie bei der serverseitigen Durchsetzung. Beim unstrukturierten Netzwerk werden Benutzerzuordnungen und administrative Privilegien deshalb in der Replikationsgruppe des Rolleneigentümers gespeichert. Im Unterschied zur serverseitigen Durchsetzung ist der Dateneigentümer nicht zwingend für die Übertragung dieser Privilegien in den Privilegienspeicher verantwortlich. Stattdessen kontrollieren die Stellvertreter die Übertragung.

3.8.6.3 Speicherung von Berechtigungen im DHT-Privilegienspeicher

Privilegien sollten immer in der Stellvertretergruppe des Dateneigentümers gespeichert werden, auch wenn der Privilegienspeicher auf einer DHT basiert. Denn auch dort werden sie zur Durchsetzung von Anfragen benötigt. Die DHT dient somit lediglich dem Auffinden der Objekteigentümer und der DGIOs ihrer Stellvertretergruppe sowie der XACML Policies.

Privilegien Die Speicherung von Privilegien findet nicht direkt in der DHT statt. Stattdessen werden für die Speicherung die Stellvertreter des Datenobjekteigentümers gewählt. Der zusätzliche Aufwand für diese zweite Replikationsverwaltung ist gering, da die Stellvertretergruppe schon existiert. Bei Gruppenänderungen muss das neue Gruppenmitglied zusätzlich lediglich die gespeicherten Daten von den bestehenden Gruppenmitgliedern abfragen und diese lokal speichern, sofern sie für gültig befunden wurden. Zum Speichern von Privilegien gelten sinngemäß die Ausführungen des unstrukturierten Netzwerks.

Benutzerzuordnungen und administrative Privilegien von Rollen Benutzerzuordnungen werden in der Stellvertretergruppe des Rolleneigentümers gespeichert. Administrativen Privilegien von Rollen werden neben der Speicherung in der Stellvertretergruppe des Datenobjekteigentümers auch in der Stellvertretergruppe des Rolleneigentümers gespeichert. Die Übertragung dieser Daten zu den Stellvertretern des Rolleneigentümers nehmen hierbei die Stellvertreter des Dateneigentümers vor.

3.8.6.4 Abfragen von Berechtigungen im Privilegienspeicher

Privilegien werden immer in der Stellvertretergruppe des Dateneigentümers abgelegt. Die Stellvertreter können bei der Durchsetzung der Privilegien ihre Entscheidung deshalb auf Basis der lokal gespeicherten Privilegien vornehmen.

Benutzerzuordnungen werden immer von der Stellvertretergruppe des Rolleneigentümers gespeichert. Für diese Stellvertreter ist deshalb die Überprüfung der Korrektheit dieser Zuordnungen anhand der lokalen Benutzerzuordnungen möglich.

Fragen andere Peers Privilegien oder Benutzerzuordnungen ab, muss die Korrektheit gespeicherter Privilegien bzw. Benutzerzuordnungen entweder durch einen umfassenden Stichprobentest mit 32 Proben oder der Überprüfung des Delegationspfads erfolgen. Da alle Stellvertreter die Privilegien bzw. Benutzerzuordnungen des Delegationspfads lokal vorliegen haben wird dieser mit dem angefragten Privileg zurückgegeben. Hierdurch kann der Anfragende Peer den Delegationspfad überprüfen und braucht deshalb anstatt der 32 Stichproben nur fünf Stichproben von den Stellvertretern.

3.8.6.5 Alternative Lösungen zur Organisation des Privilegienspeichers

Ein alternativer Lösungsansatz besteht darin, die Privilegien und Benutzerzuordnungen direkt im Privilegienspeicher zu speichern. Beim DHT-basierten Privilegienspeicher werden die Privilegien und Benutzerzuordnungen dann unter einem geeignet gewählten Schlüssel abgelegt, für unstrukturierte Netzwerke entsprechend bei der Replikationsgruppe des Grantors. In beiden Fällen ist damit Verfügbarkeit der Privilegien und Benutzerzuordnungen im Netzwerk garantiert und die Daten werden gleichmäßig über die Teilnehmer des Netzwerks verteilt. Allerdings sind diese dadurch auch schwer auffindbar. So muss bei einem unstrukturierten Netzwerk für jede Benutzerzuordnung und jedes Privileg das DGIO des Grantors gesucht werden. Bei Verwendung einer DHT muss über den Schlüssel der Berechtigung gesucht werden. Viel entscheidender ist allerdings, dass überhaupt nach den Privilegien gesucht werden muss. Bei der in PACS gewählten Organisation liegen diese schon lokal bei dem Peer vor, der die Auswertung der Privilegien vornehmen muss. Aufgrund dieses Nachteils wurde diese Alternative letztlich verworfen.

Theoretisch ist es auch möglich, ganz ohne einen separaten Privilegienspeicher auszukommen. Berechtigungen sind hierbei mit den Datenobjekten zu verknüpfen und mit diesen zu speichern. So sind sie auch noch auffindbar, wenn sich der Grantor nicht mehr im Netzwerk befindet; ihre Aktualität ist durch die Verknüpfung mit den Datenobjekten gewährleistet. Eine solche Lösung ist allerdings ungenügend, falls auch andere Benutzer berechtigt sind, Datenobjekte zu verändern. Mit jeder Datenänderung müssen die Privilegien angepasst werden, was im Normalfall nur dem Eigentümer der Daten obliegt. Deshalb ist bei einer solchen Organisation eine administrative Delegation der Privilegien nicht möglich, weshalb auch diese Variante verworfen wurde.

3.8.7 Bedeutung der clientseitigen Durchsetzung

Die clientseitige Durchsetzung ist gegenüber der serverseitigen Durchsetzung in der Auswertung von Export Policies beschränkt. Der administrative Aufwand beim clientseitigen Verfahren ist wesentlich höher. Hinzu kommt, dass durch die verschlüsselten Daten die Anfrageverarbeitung beim Datenanbieter wie sie bei PDMS Systemen vorkommt eingeschränkt ist (siehe z.B. [WL06, HILM02]). Allerdings bietet die clientseitige Durchsetzung als einzige die Möglichkeit der sicheren Datenreplikation im P2P-Umfeld. Insbesondere bleibt nur so der Schutz der Daten durch die Zugriffskontrolle gewährleistet. Clientseitige Durchsetzung ist sowohl für P2P-Netzwerke mit einer hohen Fluktuation der Teilnehmer als auch für PDMS Systeme gleichermaßen geeignet. Durch Unterstützung der administrativen Delegation und RBAC ist die Administration der Zugriffskontrollregeln auch für eine große Anzahl an Peers und Benutzer zu bewältigen. Datenreplikation ist für P2P-Netzwerke der maßgebliche Mechanismus, um die Verfügbarkeit der Daten sicherzustellen. Auf diese zu verzichten, ist deshalb nur für sehr wenige Anwendungsgebiete möglich. Für die auf Replikation angewiesenen Anwendungen ist clientseitige Durchsetzung ohne Alternative.

3.9 Zusammenfassung

Die in diesem Kapitel vorgestellte PACS-Zugriffskontrolle besitzt viele Freiheitsgrade. Zwar wurde das Zugriffskontrollmodell von PACS exakt festgelegt, aber für die Verwaltung der Privilegien wurden zwei alternative Privilegienspeicher vorgestellt, nämlich der Privilegienspeicher für das unstrukturierten Netzwerk mit aktiver Replikation und definierte Replikationsgruppe (im Folgenden unstrukturierter P2P-Privilegienspeicher) sowie der DHT-Privilegienspeicher. Des Weiteren wurden die serverseitige und clientseitige Durchsetzung vorgestellt.

PACS bietet aufgrund seines Designs die Unterstützung für rollenbasierte Zugriffskontrolle und administrative Delegation. Des Weiteren nutzt PACS die Zugriffskontrollsysteme der beteiligten Peers durch die Extraktion der Zugriffskontrollregeln in Export Policies.

Beim unstrukturierten P2P-Privilegienspeicher handelt es sich um eine komplette Eigenentwicklung. Beim DHT-Privilegienspeicher konnte auf bestehende Arbeiten aufgebaut werden aber der Transfer der Algorithmen auf Chord, die Erweiterung des Chord Protokolls und die verlässliche DHT-Anwendungen sind hierbei ein Eigenbeitrag. Die Privilegienspeicherarten unterscheiden sich in ihrer Funktionalität nicht und können deshalb nach Effizienz und Leistung ausgewählt werden.

Bei den vorgestellten Durchsetzungsvarianten ist dies nicht der Fall. Die Organisation der Privilegien für die serverseitige und clientseitige Durchsetzung wurde hierbei speziell für

PACS entwickelt. Die clientseitige Durchsetzung mit der Schlüsselverwaltung und der autonomen Verwaltung der Stellvertretergruppe ist ein eigener Beitrat der speziell für PACS entwickelt wurde. Die serverseitige Durchsetzung verursacht weniger Verwaltungsaufwand und ist gegenüber der clientseitigen Durchsetzung deshalb effizienter. Allerdings unterstützt sie keine sichere Datenreplikation. Nur für Anwendungsgebiete die auf eine solche sichere Datenreplikation nicht verzichtet können ist der Einsatz der clientseitigen Durchsetzung sinnvoll. Neben dem größeren Verwaltungsaufwand dieser Durchsetzungsvariante ist auch ihre Beschränkung bei der Auswertung von Attributen bei der XACML Evaluation zu nennen. Trotzdem unterstützt auch die clientseitige Durchsetzung das PACS-Zugriffskontrollmodell vollständig. Sie ist damit die erste clientseitige Durchsetzungsvariante die sowohl RBAC als auch administrative Delegation unterstützt.

Kapitel 4

Der PACS-Prototyp

Die prototypische Implementierung des in Kapitel 3 vorgestellten PACS-Moduls befasst sich mit den für seine Funktionsweise entscheidenden Punkten. Dies ist zum einen die Zuverlässigkeit des P2P-Netzwerkes, denn diese entscheidet über die Zuverlässigkeit des gesamten PACS-Ansatzes. Zudem stellen Kommunikationskosten in einem P2P-Umfeld die dominierende Größe dar. Sie bestimmen deshalb den Aufwand, den ein solches PACS-Modul verursacht. Diese Kommunikationskosten hängen im Wesentlichen von der Realisierung des Privilegienspeichers im P2P-Netzwerk und der Organisation der Privilegien ab. Aus diesen Gründen konzentriert sich der Prototyp auf die Realisierung einer Simulation des Privilegienspeichers.

Zur Evaluation der Kommunikationskosten und der Zuverlässigkeit der Datenspeicherung in einem P2P-Umfeld, wurde eine Simulation eines P2P-Netzwerkes erstellt (Kapitel 4.1), die die zwei vorgestellten Privilegienspeicher von PACS implementiert (Kapitel 4.2 und 4.3) und anschließend deren Kommunikationskosten ermittelt sowie deren Zuverlässigkeit überprüft.

Die anderen Bereiche des PACS-Moduls, wie die Auswertung globaler Privilegien und die Evaluation von Export Policies, wurden nicht speziell für PACS entwickelt. Für die Evaluation der in den Export Policies enthaltenen XACML Policies reicht es hierbei aus, schon vorhandene XACML Implementierungen zu verwenden, wie diejenige von Sun [Sun]. Gleiches gilt für die Implementierung der konkreten Auswertung globaler Privilegien. Diese erzielt ebenfalls keinen neuen Erkenntnisgewinn, da deren Realisierung schon durch das Modell genau spezifiziert ist. Durch die Realisierung der clientseitigen Durchsetzung von PACS kann zwar der exakte zusätzliche rechnerischen Aufwand zur Erzeugung von Schlüsseln und Ver- und Entschlüsselung von Daten bestimmt werden, dieser Aufwand ist allerdings für den hier gezeigten Fall gegenüber den Kommunikationskosten zu vernachlässigen. Auf eine Realisierung wurde deshalb verzichtet.

4.1 Simulationsumgebung

Um den Aufwand und die Zuverlässigkeit des Privilegienspeichers abschätzen zu können, wird für PACS eine Simulation eines P2P-Netzwerks erstellt. Den Vorzug erhielt die Simulation gegenüber einer realen Implementierung aus folgenden Gründen. Erstens werden für die Simulation eines P2P-Netzwerks wesentlich weniger Rechnerressourcen benötigt, als dies bei einer realen Implementierung der Fall ist. Zweitens können durch eine Simulation die Eigenschaften des P2P-Netzwerks exakter bestimmt werden. So können die Größe des Netzwerks und Veränderungen des Netzwerks in einer Simulation festgelegt werden, was in einer rea-

len Implementierung nicht ohne weiteres möglich ist. Drittens können das Verhalten bösartig kooperierender Peers und deren Anteil im Netzwerk in einer Simulation nach Bedarf leicht variiert werden. So kann z.B. auch ein Konfiguration gewählt werden, die dem Worst-Case Szenario für den Privilegienspeicher entspricht. Die Zuverlässigkeit und der Aufwand, den der Privilegienspeicher in dieser Extremsituationen erzeugt, können dadurch ermittelt werden. In einem realen Umfeld ist das Verhalten bösartig kooperierender Peers kaum nachzubilden, da deren Verhalten im Idealfall auf dem Verhalten der anderen bösartigen Peers basiert. In einer realen Implementierung ist hierfür der ständige zeitnahe Austausch zwischen den anderen kooperierenden Peers nötig. Dies macht die reale Implementierung des Verhaltens bösartig kooperierender Peers wesentlich aufwändiger.

4.1.1 Auswahl der Simulationsbibliothek

Eine Vielzahl von Simulationsbibliotheken erleichtern die Entwicklung einer P2P-Netzwerk Simulation. Die ideale Simulationsbibliothek für PACS muss folgende Voraussetzungen erfüllen:

- Die Simulationsbibliothek muss die für PACS richtige Abstraktionsebene besitzen. Für den Privilegienspeicher sind lediglich die organisatorischen Abläufe und die Information, welcher Peer an wen Nachrichten sendet, von Interesse. Für PACS ist es hingegen nicht entscheidend, die konkreten Abläufe der Netzwerkverbindung und die Details des Datentransfers zu simulieren. Deshalb ist eine Abstraktion dieser Details durch die Simulationsumgebung für PACS wünschenswert. Eine einfache Möglichkeit, diese Vorgänge mit Hilfe der Simulationsbibliothek zu realisieren, ist deshalb unabdingbar.

- Da insbesondere Netzwerke mit bösartig kooperierenden Peers simuliert werden, ist die einfache Simulation von derartigem Verhalten ebenfalls eine Voraussetzung für die Simulationsumgebung.

- Da vorgesehen ist, verschiedene P2P-Infrastrukturen miteinander zu vergleichen, ist eine Trennung von Infrastruktur und der letztendlichen Implementierung ebenfalls wünschenswert. Insbesondere muss es möglich sein, verschiedene Infrastrukturen mit derselben Simulationsbibliothek zu realisieren.

- Die Skalierbarkeit der Simulation muss ausreichend groß sein, damit zumindest Netzwerke bis zu einer Größe von zehntausenden Peers simuliert werden können.

Anhand dieser Auswahlkriterien wurde eine geeignete Simulationsbibliothek ermittelt. Ausgangspunkt für die Entscheidung war die Studie von Naicken u.a. [NBL+07, NBLR06, NBL+06], die verschiedene Simulationsumgebungen vorstellt und miteinander vergleicht. Simulationsumgebungen, die lediglich für eine spezielle Infrastruktur entwickelt wurden, wie der FreePastry Simulator [Frea] oder der Chord Simulator [Cho], schieden deshalb aus. Ebenfalls wurden paketbasierte Simulatoren (wie ns-2 [Nsn]) ausgeschlossen. Allgemeine diskrete ereignisbasierte Simulationsumgebungen, wie Javasim [Jav] oder CSIM [CSi], bieten keine spezielle Unterstützung für die Simulation von P2P-Netzwerken. Die Realisierung einer Simulation hiermit ist deshalb aufwändiger als mit Simulationsumgebungen, die speziell für die Simulation von P2P-Netzwerken erstellt wurden, wie PlanetSim [LPM+04, Plaa], PeerSim [JMJV] und Overlay Weaver [STS08]. PlanetSim erfüllte von den genannten Kandidaten die

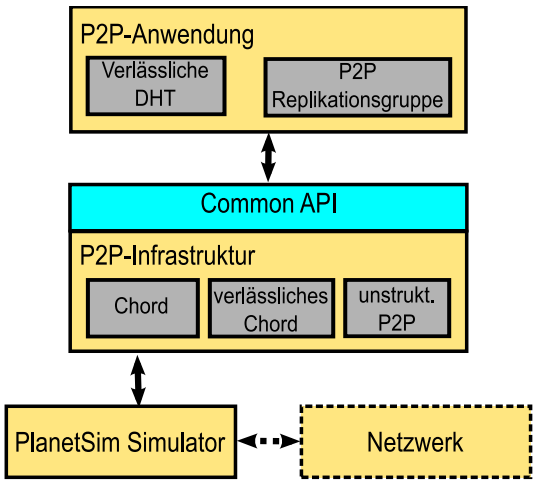


Abbildung 4.1: Architektur von PlanetSim mit PACS

Anforderungen am besten, da PlanetSim neben der detaillierten Simulation des Routingprotokolls auch die P2P-Anwendung in separaten Modulen simuliert. Zudem besaß PlanetSim zum Zeitpunkt der Evaluation als einzige Simulationsumgebung eine direkte Unterstützung bösartig kooperierender Peers.

4.1.2 Aufbau der Simulation

Die Simulationsbibliothek PlanetSim wird für die Simulation erweitert und angepasst, um das verlässliche Routing von Nachrichten in PACS realisieren zu können. Der Aufbau von PlanetSim und die für PACS vorgenommenen Erweiterungen werden in den nachfolgenden Kapiteln genauer erläutert.

4.1.2.1 Aufbau PlanetSim

Die grundlegende Architektur von PlanetSim ist in Abbildung 4.1 dargestellt. Wie ersichtlich, unterscheidet PlanetSim zwischen P2P-Anwendung und P2P-Infrastruktur. Die beiden Module kommunizieren über das Common API, das von Dabek u.a. [DZD+03] als einheitliche Schnittstelle zwischen strukturierten P2P-Netzwerken und P2P-Anwendungen vorgestellt wurde. In der Abbildung grau unterlegt, sind die in PACS für die Simulation entwickelten PlanetSim Komponenten. Nur das P2P-Infrastrukturmodul kommuniziert mit dem PlanetSim Simulator. Dieser übernimmt die Simulation der Kommunikation zwischen den Knoten des simulierten Netzwerks. Anstatt den Versand von Nachrichten zu simulieren, kann er auch den physischen Versand von Nachrichten initiieren. Dadurch ist es mit geringem Aufwand möglich, eine Simulation in eine reale P2P-Anwendung zu überführen. Diese Funktionalität ist allerdings noch nicht vollständig in der PlanetSim Simulationsbibliothek implementiert, was jedoch für die vorliegende Simulation aus den schon erläuterten Gründen nicht von Bedeutung ist.

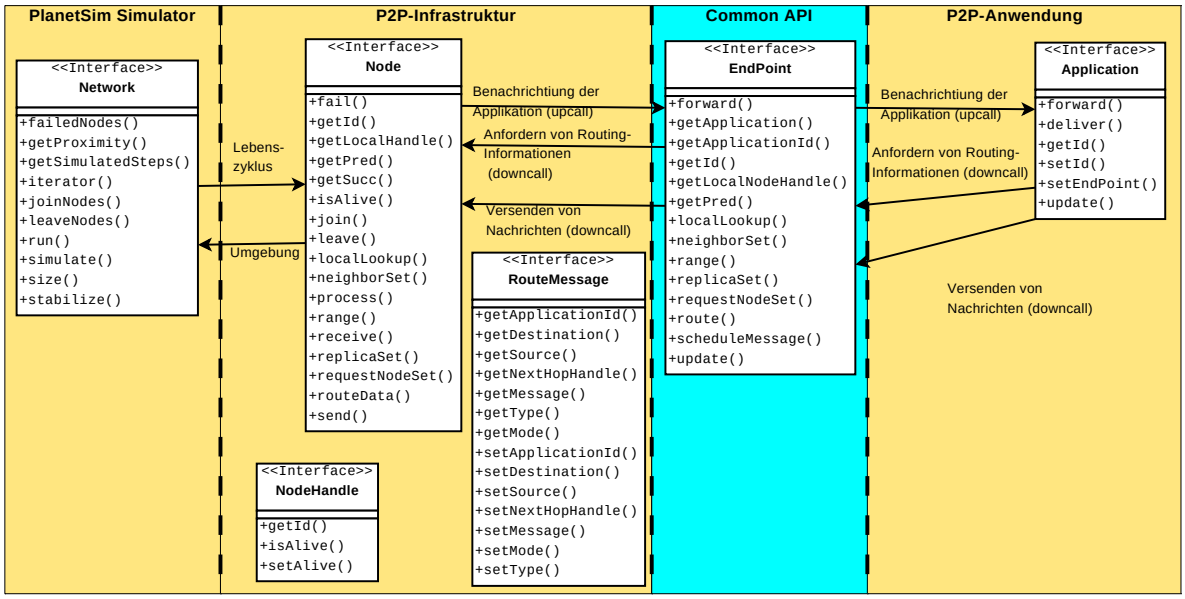


Abbildung 4.2: Wichtige Klassen von PlanetSim und ihre wichtigsten Methoden

4.1.2.2 Die wichtigsten Klassen von PlanetSim

Für PACS kann auf bereits bestehende Klassen von PlanetSim zurückgegriffen werden. Ihre für die Simulation relevanten Methoden und ihre Funktionen als Schnittstellen zwischen den PlanetSim Modulen, werden in Abbildung 4.2 in Anlehnung an [Plab] dargestellt. Zum besseren Verständnis wurden die Klassen ihren Modulen zugeordnet. Das dargestellte Network-Interface stellt die Schnittstelle zum PlanetSim Simulator dar. Die Methoden von Network-Interface werden deshalb nicht von Knoten aufgerufen, sondern vom PlanetSim Simulator. Dieser überwacht und steuert die Simulation, indem er Änderungen am Netzwerk initiiert und den Zustand des Netzwerks protokolliert.

Hauptbestandteil des Common API ist das EndPoint-Interface. In diesem findet die Vermittlung zwischen Node-Interface und dem Application-Interface statt. Das EndPoint-Interface ist hierbei eine Fassade zu den darunterliegenden Knoten der P2P-Infrastruktur, die durch das Node-Interface repräsentiert werden. Da die Aufrufe von der darüberliegenden Anwendungsschicht kommen, werden diese auch als „downcall“ bezeichnet. Die P2P-Anwendung kommuniziert in zwei Situationen mit der P2P-Infrastruktur: Beim Versenden von Nachrichten und beim Abfragen des Routingzustands eines Node-Objekts. Zum Versenden von Nachrichten durch das darunterliegende Netzwerk dient die Methode *route*. Zum Abfragen der Routinginformationen dienen die folgenden Funktionen:

replicaSet gibt die Replikationsgruppe des Knotens zurück

getId gibt den eindeutigen Bezeichner des Knotens zurück

getLocalNodeHandle gibt das NodeHandle-Objekt des Knotens zurück

getPred und getSucc geben den Predecessor bzw. Successor des Knotens zurück

range gibt den Schlüsselbereich zurück, für den der Knoten verantwortlich ist

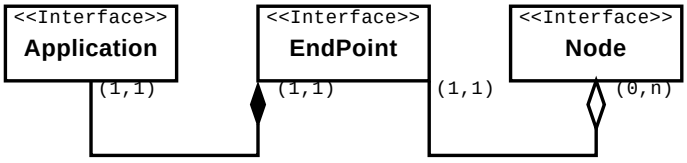


Abbildung 4.3: Wichtige Klassen eines simulierten Knotens

neighborSet gibt die Knoten in der Nachbarschaft eines Knotens zurück

Zusätzlich ist das EndPoint-Interface auch Schnittstelle zwischen P2P-Infrastruktur und P2P-Anwendung, durch die sogenannten „upcall“ Methoden. Dies sind die Methoden $forward$, $scheduleMessage$ und $update$. Sie rufen die zugehörigen Methoden im Application-Interface auf. Die entsprechende Methode für $scheduleMessage$ im Application-Interface ist hierbei $deliver$. Die Methoden werden von den Knoten der P2P-Infrastruktur zur Übermittlung von Nachrichten ($scheduleMessage$) an die P2P-Anwendung, bei Änderungen der Routinginformationen ($update$) oder der Weiterleitung von Nachrichten ($forward$) aufgerufen. Das EndPoint-Interface stellt also den Application-Objekten Methoden zum Zugriff auf Methoden und Zustand des Node-Objekts zur Verfügung. Für Node-Objekte stellt es Methoden bereit zur Benachrichtigung über Zustandsänderungen an der P2P-Infrastruktur und zur Weiterleitung von Nachrichten durch die P2P-Infrastruktur sowie zur Weitergabe von ankommenden Nachrichten an die P2P-Anwendung.

Die Implementierung der P2P-Infrastruktur erfolgt in einer Node-Klasse, während die P2P-Anwendung in einer Application-Klasse implementiert wird.

In Abbildung 4.2 ist die für PACS entwickelte Erweiterung ablesbar. Das Common API wird zur Unterstützung des verlässlichen Routings und der verlässlichen DHT geringfügig um die Methode $requestNodeSet$ im Node und EndPoint-Interface zur Anforderung einer sicheren Knotenmenge (SKM) ergänzt. Hierdurch erhält auch die P2P-Anwendung Zugriff auf redundantes Routing in der P2P-Infrastruktur.

Wichtig für das Verständnis von PlanetSim ist, dass jeder simulierte Knoten aus mehreren Application-Objekten, mehreren zugehörigen EndPoint-Objekten und einem Node-Objekt besteht. Diese Beziehungen zwischen den verschiedenen Klassen sind in Abbildung 4.3 dargestellt. In PACS wurde allerdings von der Möglichkeit kein Gebrauch gemacht, einen Knoten mehrere Applikation zuzuordnen. Stattdessen besitzt jeder Knoten immer nur ein Application-Objekt.

Die NodeHandle-Klasse enthält neben der ID des Knotens auch seine aktuelle IP-Adresse, unter der er erreichbar ist. Um eine Nachricht an einen anderen Peer senden zu können, ist ein aktuelles NodeHandle-Objekt des Peers erforderlich. Einträge oder Verweise zu anderen Knoten bestehen deshalb immer aus NodeHandle-Objekten.

4.1.2.3 Nachrichtenarten in PlanetSim

Durch die strikte Trennung zwischen P2P-Anwendung und P2P-Infrastruktur findet die Kommunikation zwischen Anwendungs- und Infrastrukturebene durch Nachrichten statt. Nachrichten in PlanetSim lassen sich dadurch in Anwendungsnachrichten und Netzwerknachrichten unterteilen.

Anwendungsnachrichten sind Nachrichten, die von der Anwendungsschicht über den Aufruf der $route$ Methode im EndPoint-Interface an das darunterliegende Netzwerk übergeben

werden. Die Nachricht wird gemäß dem von den Node-Objekten implementierten Protokoll zu ihrem angegebenen Ziel weitergeleitet. Node-Objekte brauchen hierzu keinerlei Wissen über den Inhalt der Nachricht, weshalb dieser Inhalt immer als Data-Nachricht deklariert wird.

Netzwerknachrichten sind Nachrichten, die ausschließlich zwischen den Node-Objekten einer Simulation versendet werden. Sie existieren deshalb nur auf der Ebene der P2P-Infrastruktur. Im Gegensatz zu Anwendungsnachrichten ist Node-Objekten der Aufbau dieser Nachrichten bekannt. Deshalb können Inhalt, Modus und Typ derselben deren Verarbeitung bestimmen. In PlanetSim kann eine Nachricht drei verschiedene Modi besitzen:

Request für Anfragen, die eine Antwort des Zielknotens erfordern

Reply für Antworten auf *Request* Nachrichten

Refresh für Benachrichtigungen, die keine Antwort des Zielknotens erfordern

Nachrichtentypen und Nachrichteninhalte sind spezifisch für jede P2P-Infrastruktur und P2P-Anwendung. Die bei PACS verwendeten bzw. dafür entwickelten Nachrichtentypen werden deshalb bei der Erklärung der verteilten Hashtabelle und dem unstrukturierten P2P-Netzwerk erläutert (siehe Kapiteln 4.2 und 4.3).

Jede Nachricht, die im Netzwerk versendet werden soll, wird für das Routing im Netzwerk in ein RouteMessage-Objekt eingebettet. Das RouteMessage-Objekt dient hierbei als Container für die zu versendenden Netzwerk- und Anwendungsnachrichten und verwaltet die für das Routing nötigen Daten stellvertretend für die darin enthaltene Nachricht. Die wichtigsten Attribute dieses Objekts sind:

- Sender der Nachricht, also der Knoten, der die Nachricht verschickt hat
- Ziel der Nachricht, dieses entspricht dem Schlüssel für den der Successor gesucht wird
- Nächster Knoten (next Hop), zu dem die Nachricht geroutet wird

Beim Routing der Nachricht durch die P2P-Infrastruktur wird der Successor für den Schlüssel der Nachricht gesucht und deshalb an den durch das Routingprotokoll festgelegten nächsten Knoten weitergeleitet. Werden Nachrichten direkt an einen Knoten versendet, entspricht das Ziel des RouteMessage-Objekts dem darin ebenfalls festgelegten nächsten Knoten. Der Ablauf einer Nachrichtenverarbeitung in PlanetSim erfolgt immer gemäß dem gleichen Schema. Dieser wird nachfolgend erläutert.

4.1.2.4 Der Nachrichtenfluss in PlanetSim

Netzwerknachrichten werden, nachdem sie in ein RouteMessage-Objekt eingefügt wurden, durch Aufrufe der *send* Methode des Node-Objekts zu anderen Knoten verschickt. *send* legt die Nachricht hierbei lediglich in einer Warteschlange für ausgehende Nachrichten ab. Die Versendung der Nachricht, und somit ihre Übermittlung an den Empfänger, übernimmt das Netzwerk. Es greift hierbei auf die Warteschlange mit ausgehenden Nachrichten über die Methode $outMessages$ des Node-Objektes zu und übermittelt diese an deren angegebene nächsten Knoten. Zur Übergabe der Nachricht beim Empfänger wird bei dessen Node-Objekt die *receive* Methode mit der Nachricht als Übergabeparameter aufgerufen. Da die Versendung von Nachrichten in PACS lediglich simuliert wird, übernimmt der PlanetSim Simulator diese Aufgabe. Bei einem physischen Versand bedienen sich die Netzwerkklassen, die sich um die

ein- und ausgehenden Nachrichten kümmern, ebenfalls der beiden Methoden *receive* und $outMessages$.

Bei Anwendungsnachrichten unterscheidet sich das Vorgehen etwas. Das Versenden von Nachrichten durch die Anwendung geschieht immer über *route* Aufrufe beim zur Anwendung gehörenden EndPoint-Objekt. Anwendungsnachrichten werden hierbei als Data-Nachricht in ein RouteMessage-Objekt verpackt und können so über das P2P-Netzwerk verschickt werden. D.h., Anwendungsnachrichten werden über den selben Mechanismus wie Netzwerknachrichten versendet. Node-Objekte können den Inhalt der Anwendungsnachrichten nicht interpretieren. Lediglich Informationen des RouteMessage-Objektes, das die Anwendungsnachricht enthält, sind für das Node-Objekt einsehbar und entsprechend auswertbar und manipulierbar. Damit auch der Nachrichteninhalt auf das Weiterleiten der Nachricht Einfluss nehmen kann, ist die $forward$ Methode der EndPoint-Objekte vorgesehen. Mit dieser wird dem Application-Objekt die zurzeit bearbeitete Anwendungsnachricht übergeben. Die Anwendung kann dadurch die Nachricht entsprechend verändern oder auch ihre Weiterleitung durch den Knoten ganz unterbinden.

Kommt eine Nachricht bei ihrem Zielknoten an, wird diese vom Node-Objekt nicht mehr weitergeleitet, sondern an die Anwendung über die $scheduleMessage$ Methode des EndPoint-Objekts weitergegeben. Sodann ist der Nachrichtenfluss einer Anwendungsnachricht beendet.

4.1.2.5 Ablauf der Simulation

Da das Netzwerk simuliert wird, kann dieses durch den PlanetSim Simulator genau bestimmt werden. Die Simulation läuft hierbei in virtuellen Zeitschritten ab, die auch als Simulationsschritte bezeichnet werden. Zu jedem Simulationsschritt können Änderungen am Netzwerk vorgenommen werden, indem Knoten dem Netzwerk beitreten (join), es verlassen (leave) oder ausfallen (fail). Zudem wird durch den PlanetSim Simulator der Versand von Nachrichten simuliert. Hierzu werden bei jedem Simulationsschritt die durch $outMessages$ ermittelten ausgehenden Nachrichten eines Knotens an die entsprechenden Empfänger der Nachricht mittels *receive* übermittelt. Anschließend wird für jeden Simulationsknoten die Verarbeitung dieser Nachrichten über den Aufruf seiner *process* Methode angestoßen.

Das Verhalten des PlanetSim Simulators wie auch die gesamte Simulation kann über eine Konfigurationsdatei bestimmt werden. Insbesondere kann hier festgelegt werden, wie viele bösartige Knoten im Netzwerk existieren und wie viele davon miteinander kooperieren. Wichtig ist noch zu erwähnen, dass das Verhalten der Simulation an verschiedenen Stellen durch die Erzeugung von Zufallszahlen bestimmt wird. Vor allem für das Testen der Implementierung ist es nötig, einen bestimmten Zustand einer Simulation wiederholt erzeugen zu können. Hierdurch können auch Vergleiche zwischen dem Verhalten verschiedener Alternativen exakter durchgeführt werden. Um dies zu erreichen wurde die PlanetSim Simulationsbibliothek so verändert, dass Zufallszahlen der Simulation immer über einen globalen Zufallszahlengenerator *RandomGenerator* generiert werden. *RandomGenerator* wurde hierbei als Singelton Entwicklungsmuster realisiert. Sein Startzustand am Beginn der Simulation wird durch eine „Seed“ festgelegt, die aus einer Konfigurationsdatei ausgelesen wird. Durch dieses Vorgehen ist es möglich, den Ablauf einer Simulation exakt wiederholen zu können, da der Zufallszahlengenerator wiederum dieselbe Sequenz an Zufallszahlen erzeugt, wenn er mit der gleichen Seed erzeugt wurde. Die für den Ablauf der Simulation wichtigsten Parameter sind in Tabelle A.1 im Anhang aufgelistet.

4.1.2.6 Korrektheit der Implementierung

Um die Korrektheit der Simulationsergebnisse der Implementierung zu gewährleisten, müssen Sicherheitsvorkehrungen getroffen werden.

Die PlanetSim Bibliothek nimmt eine umfassende und detaillierte Simulation des P2P-Netzwerks vor. Insbesondere durch die Simulation einzelner Nachrichten ergibt sich für P2P-Infrastrukturen und P2P Anwendungen keinerlei Vereinfachung in der Kommunikation mit den anderen-Knoten. Lediglich das konkrete Versenden von Nachrichten wird durch die Verwendung des PlanetSim Simulators vereinfacht, was aber außerhalb des Aufgabengebiets der P2P-Infrastruktur und P2P-Anwendung liegt. Die Komplexität einer realen Implementierung ist hiermit gegeben.

Durch die isolierte Modellierung und Instantiierung jedes einzelnen Knotens in der Simulation besteht keinerlei direkte Verbindung zwischen den Knoten. Der Informationsaustausch zwischen den Knoten erfolgt dadurch immer asynchron durch den Austausch von Nachrichten, so wie dies auch in realen Netzwerken der Fall ist. Eine Beeinflussung der Algorithmen durch globales Wissen kann deshalb nicht aus Versehen geschehen, sondern muss explizit vorgenommen werden. Zudem sind die modellierten Knoten vom Netzwerk unabhängig. Sie benötigen dieses lediglich für den Austausch von Nachrichten. Deshalb ist es möglich, den simulierten Nachrichtenaustausch durch den PlanetSim Simulator durch einen realen Austausch der Nachrichten zu ersetzen. Die Verwendung von in der Realität nicht vorhandenen Informationen wird durch die Architektur von PlanetSim so verhindert.

Neben umfassenden JUnit Tests [JUn], die den korrekten Ablauf der Protokolle und der Simulationsumgebung als Ganzes überprüfen, werden zudem sofern möglich Vergleichsmessungen vorgenommen, um auch auf diese Art einen Nachweis zu führen, dass die Ergebnisse korrekt sind.

4.2 Implementierung der verteilten Hashtabelle

Die für PACS entwickelte verteilte Hashtabelle (verlässliche DHT) (siehe Kapitel 3.5.7.8) wird nun in die PlanetSim Simulationsbibliothek implementiert. Sie basiert auf dem für PACS erstellten verlässlichen Chord Protokoll. Da die vom Autor entwickelten Algorithmen für das Routing und die Verwaltung der Routinginfrastruktur bereits in Kapitel 3.5.7 erläutert wurden, finden hier lediglich die Besonderheiten der Implementierung Beachtung: Die von der Simulation versendeten Nachrichten, sowie die wichtigsten Parameter der Simulation und deren Einstellungen.

4.2.1 Verlässliches Chord Protokoll

Für das Chord Protokoll existierte bereits eine Implementierung für PlanetSim. Da diese jedoch schlecht strukturiert war und die für das verlässliche Chord Protokoll nötigen Erweiterungen nicht integriert werden konnten, wurde eine verbesserte Implementierung nötig. Die von der Chord Implementierung von PlanetSim verwendeten Nachrichtentypen wurden dabei weitestgehend übernommen.

4.2.1.1 Nachrichten des Chord Protokolls

Die Simulation des Chord Protokolls unterscheidet folgende Nachrichtentypen:

FindSucc Eine FindSucc-Nachricht wird versendet, wenn es darum geht, den Successor-Knoten für eine ChordID ausfindig zu machen. Dies geschieht üblicherweise durch das Weiterleiten dieser Nachricht gemäß dem Chord Protokoll. Versendet werden diese Nachrichten beim Eintritt eines Knotens in das Netzwerk und beim Ermitteln der Fingertabelleneinträge bzw. deren Überprüfung oder Korrektur.

SetSucc Eine SetSucc-Nachricht teilt dem Predecessor eines Knotens seinen neuen Successor mit. Eine solche Nachricht wird niemals geroutet, sondern immer direkt an den Predecessor gesendet. Da der Sender dieser Nachricht keine Antwort erwartet, befindet sich diese im Modus Refresh. Die SetSucc-Nachricht wird von einem Knoten generiert, wenn er das Netzwerk vorschriftsgemäß verlässt. Beim Ausfall eines Peers hingegen unterbleibt die Versendung einer solchen Nachricht.

SetPre Eine SetPre-Nachricht sendet ein Knoten an seinen Successor, um diesen über einen neuen Predecessor zu informieren. Es handelt sich um eine direkte Nachricht, die den Modus Refresh besitzt. Wie die SetSucc-Nachricht wird auch die SetPre-Nachricht vom einem Knoten generiert, der das Netzwerk vorschriftsgemäß verlässt und sich hierzu bei seinem Successor abmeldet.

GetPre Eine GetPre-Nachricht fordert den Peer, an den die Nachricht gesendet wird dazu auf, seinen Predecessor zurückzusenden. Sie wird immer direkt an den Empfänger (im Normalfall der Successor des Knotens) adressiert und deshalb nie über das Chord Protokoll geroutet. Die Nachricht markiert den Start des Stabilisierungsprozesses bei Chord. Dieser dient der Überprüfung der Successor-Liste sowie der gespeicherten Routinginformationen.

Notify Eine Notify-Nachricht ist Teil des Stabilisierungsprozesses. Mit dieser bestätigt der Peer, der diesen Prozess initiiert hat, dem Successor, dass er Successor des Peers ist. Da die Nachricht keine Antwort erfordert, besitzt sie den Modus Refresh.

SuccList Eine SuccList-Nachricht ist der Abschluss des Stabilisierungsprozesses. Der nun bestätigte Successor wird aufgefordert seine Successor-Liste zu übermitteln. Auch bei dieser Nachricht handelt es sich um eine direkte Nachricht. Nach Erhalt der Antwort durch den Successor wird die eigene Successor-Liste mit der Successor-Liste des Successor verglichen und gegebenenfalls die eigene Successor-Liste aktualisiert.

Data Alle Anwendungsnachrichten haben auf der P2P-Infrastrukturebene den Typ Data-Nachricht. Sie werden lediglich durch das Chord Protokoll geroutet. Ihre Verarbeitung findet wie schon beschrieben in der P2P-Anwendung statt.

4.2.1.2 Nachrichten des verlässlichen Chord Protokolls

Für die in PACS vorgenommen Erweiterungen zum verlässlichen Chord Protokoll kommen noch folgende Nachrichtentypen hinzu:

SecureRouting Alle Nachrichten, die im Zusammenhang mit redundantem Routing versendet werden, besitzen diesen Typ. Hierzu gehören die folgenden Nachrichten:

FindNeighbour FindNeighbour-Nachrichten werden durch das Netzwerk geroutet, um Nachbarn der gesuchten ChordID zu finden. Sie besitzen den Modus Request oder Reply.

NodeList NodeList-Nachrichten sind direkte Nachrichten. Sie dienen dem Abgleich der Resultatmenge $\mathcal{M}$ mit den Replikationsgruppen ihrer Knoten. Die Knoten der Resultatmenge antworten entsprechend mit neuen Kandidaten für die Resultatmenge $\mathcal{M}$.

Forward Forward-Nachrichten dienen der Überprüfung neuer Kandidaten für die Resultatmenge $\mathcal{M}$. Auch hier handelt es sich um direkte Nachrichten mit dem Modus Request oder Reply.

Weiterhin sind noch Nachrichten unter diesem Typ registriert, die allerdings nicht mit redundantem Routing im Zusammenhang stehen, sondern für andere Erweiterungen benötigt werden.

Alive Direkte Alive-Nachrichten werden benutzt um zu überprüfen, ob sich ein Peer noch im Netzwerk befindet. Entsprechend besitzt die Nachricht den Modus Request oder Reply.

GetSucc GetSucc-Nachrichten werden für den Stichprobentest bei neuen Fingereinträgen benötigt. Auch hierbei handelt es sich um eine direkte Nachricht mit Modus Request oder Reply.

NodeSet Diese Nachrichten werden zum Anfordern einer SKM an die Bootstrap-Knoten während des Eintritts eines Knotens in das Netzwerk versendet. Es handelt sich dabei immer um direkte Nachrichten mit dem Modus Request oder Reply.

PredList Nachrichten dieses Typs repräsentieren die durch die Erweiterung von Chord um eine Predecessor-Liste nötig gewordene Entsprechung der SuccList-Nachricht. Hierzu wurde der Stabilisierungsprozess erweitert. Neben der Anforderung der Successor-Liste wird nun zusätzlich eine PredList-Nachricht an den Predecessor gesendet, die ihn auffordert, seine Predecessor-Liste zurückzusenden. Nach Erhalt der Antwort vergleicht und aktualisiert der Peer gegebenenfalls seine eigene Predecessor-Liste mit der Antwort.

4.2.1.3 Parameter für das verlässliche Chord Protokoll

Für das Chord Protokoll und seine Erweiterungen gibt es eine Reihe von Einstellungsmöglichkeiten, die über Parameter in einer Konfigurationsdatei für die Simulation festgelegt werden können. Die wichtigsten Parameter für Chord sind in Tabelle A.2 im Anhang dargestellt. Für die Erweiterungen von Chord zum verlässlichen Chord Protokoll wurden neue Parameter eingeführt, die in Tabelle A.3 des Anhangs erläutert werden.

4.2.2 Verlässliche DHT

Obwohl Anwendungsnachrichten auf P2P-Infrastrukturebene immer Data-Nachrichten sind, besitzen diese auf der P2P-Anwendungsebene ebenfalls verschiedene Nachrichtentypen und verschiedene Modi. Diese sollen an dieser Stelle genauer erläutert werden.

Insert Durch eine Insert-Nachricht veranlasst eine Anwendung das Einfügen eines Datenobjekts. Das Datenobjekt wird hierbei durch Chord beim Successor-Knoten des Datenobjektes gespeichert. Eine Insert-Nachricht kann die Modi Request und Reply besitzen. Die Beantwortung der Insert-Nachricht dient hierbei der Fehlerkontrolle.

InsertReplica InsertReplica-Nachrichten sind im Gegensatz zu den Insert Nachrichten direkte Nachrichten. D.h., sie werden nicht durch Chord geroutet, da es sich hierbei um direkte Aufforderungen an einen Peer handelt. Durch eine solche Nachricht wird der Empfänger aufgefordert das mitgelieferte Datenobjekt zu speichern. Im Normalfall erfolgt dies zum Speichern von Replikaten. InsertReplica-Nachrichten werden vom Peer, an den die Aufforderung gesendet wurde, nicht beantwortet, weshalb es nur Request-Nachrichten gibt.

Lookup Lookup-Nachrichten dienen dem Abruf von Datenobjekten. Sie werden durch das darunterliegende Chord Protokoll zum Successor der gesuchten ChordID geroutet, der dann die Objekte zurücksendet, die den angeforderten Schlüssel besitzen.

SamplingLookup Diese Nachrichten dienen der Überprüfung der Lookup-Antwort. Hierbei werden Knoten der AKM direkt nach Objekten mit dem entsprechenden Schlüssel angefragt, die diese dann beantworten.

SecureLookup Secure-Lookup Nachrichten werden an die Knoten der SKM versendet. Es handelt sich hierbei um direkte Lookup-Abfragen. Diese Nachrichten werden nicht geroutet; die angefragten Knoten liefern die Resultate für den angegebenen Schlüssel zurück. Die SKM muss zuvor durch den entsprechenden Aufruf der $requestNodeSet$ Methode ermittelt werden.

ObjektList ObjektList-Nachrichten sind ebenfalls direkte Nachrichten. Sie werden zur Ermittlung der zu speichernden Replikate und Daten beim Eintritt des Knotens in das Netzwerk benötigt. Beim Beitritt sendet ein Knoten ObjektList-Nachrichten an seine nächsten Successor-Knoten. Diese antworten mit einer Liste der von ihnen gespeicherten Objekte. Anhand dieser zurückerhaltenen Objekte kann der Knoten dann die noch fehlenden Datenobjekte ermitteln.

4.3 Implementierung des unstrukturierten P2P-Netzwerks

Das in PACS entwickelte unstrukturierte P2P-Netzwerk zusammen mit der Anwendung zur aktiven Replikation mit definierter Replikationsgruppe wird ebenfalls mit der PlanetSim Simulationsbibliothek realisiert. Die PlanetSim Simulationsbibliothek ist flexibel genug, auch unstrukturierte P2P-Netzwerke zu simulieren. Allerdings können dann nicht alle Methoden des Common API unterstützt werden, bzw. viele Methoden sind für ein unstrukturiertes Netzwerk sinnlos wie z.B. *range*, *neighborSet* und *replicaSet*. Als Folge davon können P2P-Anwendungen, die diese Methoden des Common API benutzen, nicht mit dem simulierten unstrukturierten P2P-Netzwerk kombiniert werden, das für PACS entwickelt wurde. Für das unstrukturierte P2P-Netzwerk wird deshalb die eigens entwickelte P2P Anwendung mit aktiver Replikation und definierter Replikationsgruppe implementiert. Die Simulation dieser Anwendung wird neben der Simulation des unstrukturierten P2P-Netzwerks nachfolgend vorgestellt.

4.3.1 Knoten im unstrukturierten P2P-Netzwerk

Da die grundlegenden Algorithmen schon in Kapitel 3.5.6 erläutert wurden, werden hier lediglich die wichtigsten Nachrichten und Einstellungsmöglichkeiten der Simulation vorgestellt. Im unstrukturierten P2P-Netzwerk für PACS gibt es folgende Nachrichten:

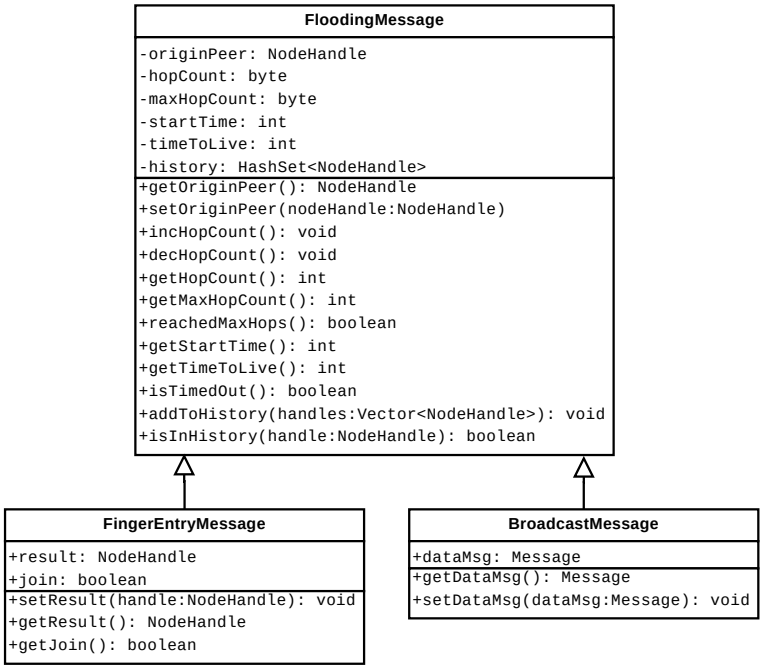


Abbildung 4.4: Klassendiagramm der Flooding-Nachrichten

FingerEntry Diese Nachricht dient dem Finden neuer Fingereinträge. Beim Beitritt eines Knotens in das Netzwerk werden diese Nachrichten per Flooding im Netzwerk verbreitet. Wird nur ein Ersatz für einen nicht mehr gültigen Fingereintrag gesucht, findet hingegen kein Flooding statt. Die Nachrichten treten in den Modi Request und Reply auf.

Alive Alive-Nachrichten dienen dem Überprüfen von Fingereinträgen und werden direkt zwischen zwei Knoten ausgetauscht. Sie werden nicht per Flooding im Netzwerk verbreitet. Alive Nachrichten besitzen den Modus Request oder Reply.

Data Anwendungsnachrichten, die direkt zwischen Knoten ausgetauscht werden, haben auf der P2P-Infrastrukturebene den Typ Data-Nachricht. Sie besitzen den Modus Request oder Reply und ihre Verarbeitung findet durch die P2P-Anwendung statt.

Broadcast Nicht direkte Anwendungsnachrichten werden durch Flooding im Netzwerk verbreitet (geroutet). Diese besitzen hierzu den Typ Broadcast-Nachrichten. Sie werden durch die P2P-Anwendung verarbeitet und besitzen den Modus Reply oder Request.

Im P2P-Netzwerk werden nur FingerEntry-Nachrichten und Broadcast-Nachrichten über Flooding im Netzwerk verbreitet. Wie im Klassendiagramm in Abbildung 4.4 ersichtlich, handelt es sich bei beiden um Flooding-Nachrichten. Der Flooding Routingalgorithmus muss nicht zwischen den beiden Nachrichtentypen unterscheiden, da die FloodingMessage-Klasse die zur Verwaltung und Kontrolle dieses Floodingprozesses notwendigen Attribute besitzt. Dies sind:

hopCount Dieses Attribut dient als Zähler der schon vorgenommenen Routingschritte dieser Nachricht.

maxHopCount maxHopCount enthält die maximale Anzahl an Routingschritten, die diese

Nachricht tätigen soll. Wenn *hopCount* diese Zahl erreicht hat, wird die Nachricht nicht mehr per Flooding im Netzwerk verteilt.

startTime Die startTime entspricht dem Zeitpunkt, zu dem die Anfrage , gestartet wurde, zu der eine Nachricht gehört.

timeToLive Diese Angabe enthält die maximale Laufzeit der Anfrage, zu der eine Nachricht gehört. Nach Ablauf dieser Lebenszeit wird die Nachricht nicht mehr weitergeleitet.

history In dieses Attribut werden die Knoten aufgenommen, an die eine Nachricht bzw. Kopien davon schon weitergeleitet wurden.

FingerEntryMessage-Objekte besitzen zusätzlich ein *result* Attribut zur Aufnahme des Resultats der FingerEntry-Anfrage. BroadcastMessage-Objekte enthalten die Anwendungsnachricht, die sie im Netzwerk verteilen. Diese wird in das $dataMsg$ Attribut abgelegt.

4.3.2 Die P2P-Anwendung

In der P2P-Anwendung findet neben der Speicherung und Verwaltung von Daten und deren Replikate auch die Verwaltung von Replikationsgruppen statt. Die Algorithmen hierzu werden in Kapitel 3.5.6 beschrieben, weshalb hier der Schwerpunkt auf den verschiedenen Nachrichtentypen der P2P-Anwendung liegt:

Insert Insert-Nachrichten sind direkte Nachrichten zwischen zwei Knoten des Netzwerks. Entsprechend besitzen sie die Modi Request und Reply.

Lookup Lookup-Nachrichten werden immer im Netzwerk durch Flooding geroutet. Sie beinhalten den zu suchenden Schlüssel oder Peer. Diese Nachrichten besitzen den Modus Request oder Reply.

Alive Alive-Nachrichten dienen dem Verwalten von Replikationsgruppen. Hierdurch wird die Erreichbarkeit der Mitglieder überprüft. Bei Gruppenänderungen dienen diese Nachrichten auch dazu, die Erreichbarkeit neuer Gruppenmitglieder und die Abwesenheit zu ersetzender Gruppenmitglieder zu bestätigen. Alive-Nachrichten sind immer direkte Nachrichten, die den Modus Request oder Reply besitzen.

Signature Mit dieser Nachricht werden Teilsignaturen von Knoten angefordert. Diese werden zum Bestätigen von Gruppenänderungen benötigt. Entsprechend handelt es sich hierbei um direkte Nachrichten, die den Modus Request oder Reply besitzen.

Share Share-Nachrichten sind direkte Nachrichten, die bei Gruppenänderungen zur Übermittlung der Teilteilschlüssel und bei Gruppenänderungen zur Übertragung der Teilschlüssel dienen. Da für diese Nachrichten keine Antwort des Empfängers notwendig ist, sind sie im Modus Request.

4.3.3 Parameter der Simulation für unstrukturierte Netzwerke

Die Simulation für unstrukturierte Netzwerke enthält eine Reihe von Parametern, die ihr Verhalten beeinflussen. Einige davon ähneln den Parametern der verlässlichen DHT, andere hingegen sind spezifisch für das Flooding von Nachrichten und die Verwaltung von Replikationsgruppen. Die wichtigsten Parameter für das unstrukturierte P2P-Netzwerk mit definierten Replikationsgruppen sind in Anhang in der Tabellen A.4 zusammengefasst.

4.4 Einflussfaktoren der Arbeitslast

Die Arbeitslast in PACS wird von zwei absoluten Faktoren bestimmt. Die *Veränderung des Netzwerks* bestimmt, wie viele Knoten in einer festgelegten Zeitperiode das Netzwerk verlassen (leave), ausfallen (fail) und dem Netzwerk beitreten (join). Um die Messungen möglichst wenig durch die variierende Größe des Netzwerks zu beeinflussen, wird jeweils sichergestellt, dass nach Ablauf der Zeitperiode genauso viele Knoten das Netzwerk verlassen haben oder ausgefallen sind, wie neue Knoten dem Netzwerk beigetreten sind. Ebenfalls wird darauf geachtet, dass der Anteil bösartiger und bösartig kooperierender Peers bei den Änderungen im Netzwerk beibehalten wird. Die Verteilung der Veränderungen (leave, fail, join) bzw. deren Reihenfolge wird hierbei zufällig festgelegt. Die Knoten, die diese Veränderung verursachen, werden zufällig ausgewählt. Knoten die einem Netzwerk beitreten, sind immer neue Knoten. D.h. ein Knoten der ein Netzwerk verlassen hat, wird diesem während der Simulation nicht mehr beitreten.

Der zweite Faktor ist die *Anzahl an Anfragen*, die von den Knoten in einer bestimmten Zeitperiode bearbeitet werden. Ein anfragender Peer darf für die Messungen das Netzwerk während der Anfrage nicht verlassen, da dies das Ergebnis verfälscht. Die Anfragen werden hierbei immer gleichmäßig über die Teilnehmer verteilt. Generell werden unter den zuvor genannten Einschränkungen immer zufällig Knoten zur Durchführung von Anfragen ausgewählt. Auch die hierbei zu speichernden Datenobjekte, insbesondere deren Schlüssel, werden zufällig ausgewählt, wobei auf eine gleichmäßige Verteilung dieser generierten Schüssel auf den Schlüsselraum geachtet wird.

Beide betrachteten Faktoren der Arbeitslast werden in der Simulation über Parameter beeinflusst, deren Werte aus einer Konfigurationsdatei ausgelesen werden. Diese Parameter sind in Tabelle A.5 im Anhang dargestellt und erläutert. Die so erzeugte Arbeitslast ist Grundlage für den Vergleich bzw. die Diskussion der Messungen in Kapitel 5.

Kapitel 5

Evaluation und Diskussion

In diesem Kapitel werden die Ergebnisse der Simulation des P2P-Netzwerks präsentiert. Zunächst werden in Kapitel 5.1 der Versuchsaufbau und die Resultate der verteilten Hashtabelle (DHT) mit ihren verschiedenen Varianten und Bestandteilen vorgestellt. Neben den Ergebnissen des Standard Chord Protokolls und des erweiterten Chord Protokolls, werden dort auch die Resultate des verlässlichen Chord Protokolls und der für PACS entwickelten verlässlichen DHT dargestellt.

Anschließend folgen die Ergebnisse des unstrukturierten P2P-Netzwerks mit aktiver Replikation für PACS (im Folgenden unstrukturierter P2P-Speicher genannt) in Kapitel 5.2. In Kapitel 5.3 erfolgt ein Vergleich der verlässlichen DHT mit dem unstrukturierten P2P-Speicher. Die Komplexitätsanalyse der für PACS entwickelten clientseitigen Durchsetzung wird in Kapitel 5.4 vorgestellt und mit der ausschließlich gruppenbasierten bzw. clientseitigen Durchsetzung verglichen. Am Ende folgt eine abschließende Bewertung des DHT- und P2P-Privilegienspeichers.

5.1 Evaluation der Messungen zur verteilten Hashtabelle

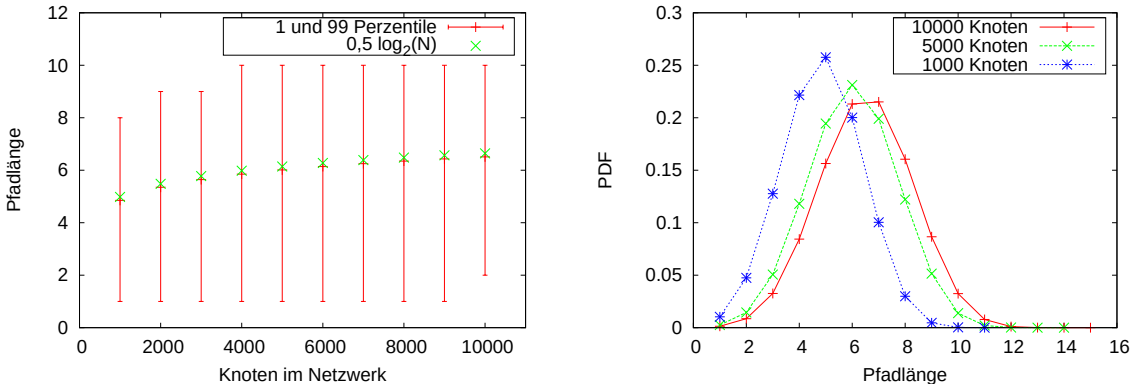
Die verteilte Hashtabelle besteht aus der P2P-Anwendung, die die DHT-Funktionalität bereitstellt, und dem der DHT zugrundeliegenden Chord Netzwerk. Es werden zunächst die Ergebnisse des Chord Netzwerks besprochen, um danach auf die DHT-Anwendung einzugehen.

5.1.1 Chord

Das Standard Chord Protokoll ist im hier betrachteten Bedrohungsszenario mit bis zu 30% bösartig kooperierenden Peers im Netzwerk nicht mehr funktionstüchtig. Dies wird durch eine Simulation belegt, bei der sämtliche für PACS vorgenommene Erweiterungen an Chord deaktiviert werden.

5.1.1.1 Routing des Chord Protokolls im statischen Netzwerk

Die Simulationen zum Standard Chord Protokolls im statischen Netzwerk belegen die Funktionstüchtigkeit des Chord Protokolls und dienen zudem als Nachweis der Korrektheit der Chord Protokoll Implementierung. Für das Chord Protokoll ist charakteristisch, dass der Successor zuverlässig in $\frac{1}{2}\log_2(N)$ Routingschritten gefunden wird, wobei N der Größe des



(a) Pfadlänge

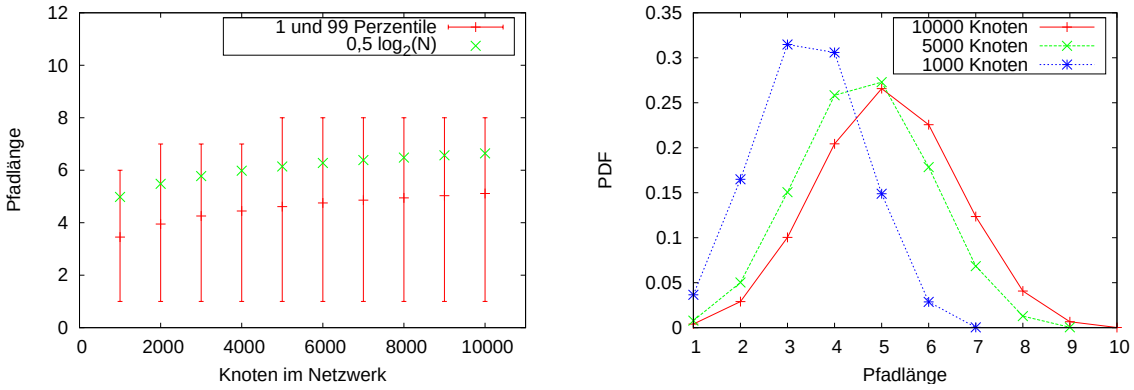
(b) Verteilung Pfadlänge

Abbildung 5.1: Pfadlängen Chord Protokoll im statischen Netzwerk

Netzwerks entspricht. Um dies mit der Simulation zu bestätigen, wird ein statisches Netzwerk erzeugt, das ausschließlich aus gutartigen Peers besteht. Für die Messung senden zufällig ausgewählte Knoten FindSucc-Nachrichten an zufällig ausgewählten Zielknoten. Der Erfolg der FindSucc-Anfrage sowie die hierzu benötigten Routingschritte werden protokolliert. Das Verhalten des Standard Chord Protokolls wird erreicht, indem die Größe der Successor-Liste (CHORD_ROUTING_SUCCESSOR_LIST_SIZE) auf null gesetzt wird. Durch diese Maßnahme wird die Successor-Liste während des Routings ignoriert (siehe Abbildung 3.21), da $routingSuccListMax$ hierdurch ebenfalls den Wert null annimmt.

Für die Ermittlung der Ergebnisse werden jeweils bis zu neun Millionen solcher Anfragen ausgewählt. Bis zu 3000 Knoten liegt deshalb eine vollständige Erfassung der Pfadlänge vor, d.h., für jeden Knoten wird die Pfadlänge zu jedem anderen Knoten im Netzwerk gemessen. In größeren Netzwerken wird für jeden Knoten nur eine zufällige Auswahl von Pfadlängen zu anderen Knoten im Netzwerk erfasst. Die Anzahl der Messungen, die jeder Knoten ausführt, ist hierbei gleichmäßig über die Knoten verteilt. Zudem wird die Messung mit zehn verschiedenen „Seeds“ des Zufallszahlengenerators wiederholt, wodurch die Zufallszahlenerzeugung variiert wird. Bei den hier dargestellten Resultaten handelt es sich um die über diese Messungen berechneten Durchschnittswerte. Die Simulationen werden für verschiedene Netzwerkgrößen durchgeführt. Die Ergebnisse dieser Simulationen sind in Abbildung 5.1 dargestellt.

Wie in Abbildung 5.1(a) zu sehen ist, ist die Pfadlänge im implementierten Chord Protokoll leicht kürzer, als dies durch die mathematische Berechnung zu erwarten war. Dargestellt sind hier neben der durchschnittlichen Pfadlänge zusätzlich jeweils die 1 und 99 Perzentile. Die Ergebnisse entsprechen den in Stoica u.a. [SMK+01] publizierten Werten. Selbiges gilt für die Verteilung der Pfadlänge, die in Abbildung 5.1(b) dargestellt ist. Wie dort zu sehen, verschiebt sich diese Verteilung mit zunehmender Netzwerkgröße weiter nach rechts, wobei sie stetig flacher wird. Aufgrund der Übereinstimmung der Messergebnisse mit den publizierten Werten für das Chord Protokoll ist die korrekte Arbeitsweise der eigenen Chord Implementierung hiermit nachgewiesen.



(a) Pfadlänge (b) Verteilung Pfadlänge

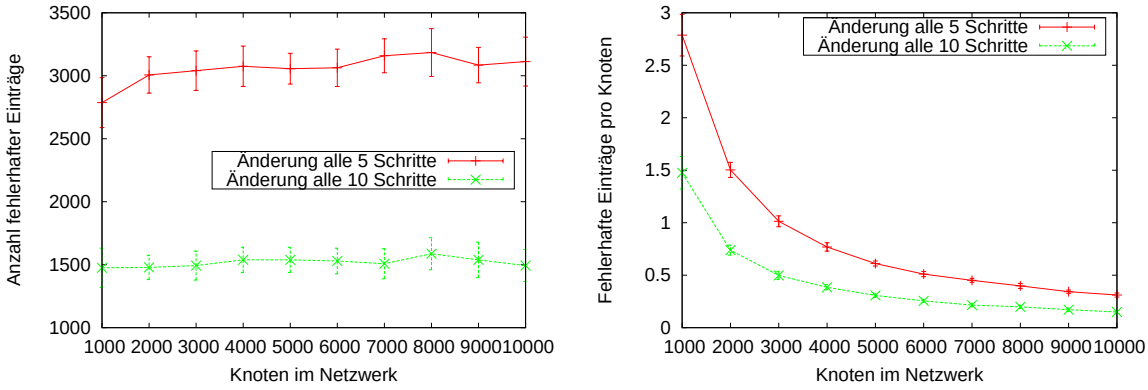
Abbildung 5.2: Pfadlängen erweitertes Chord Protokoll, statisches Netzwerk

5.1.1.2 Routing erweitertes Chord Protokoll im statischen Netzwerk

Die für PACS entwickelte Erweiterung des Chord Protokolls, vorgestellt in Kapitel 3.5.7.6, reduziert die Anzahl der Routing Schritte. Der bei dieser Erweiterung für das Routing genutzte Teil der Successor-Liste wird hierbei auf 32 beschränkt. Dieser Teil der Successor Liste wird in Chord auch für die Replikation verwendet und deshalb häufiger überprüft. Dadurch ist eine höhere Aktualität gewährleistet. Die Erweiterung erhöht die Genauigkeit der Auswahl des nächsten Predecessors und damit auch die Effizienz des Routingvorgangs. Dies lässt sich anhand der Simulationsergebnisse zeigen. Der Messaufbau für das Chord Protokoll wird hierfür nicht verändert, lediglich die beschriebene Erweiterung wird aktiviert. In Abbildung 5.2(a) wird hierbei zum Vergleich die mathematisch berechnete Anzahl an Routingschritten dargestellt. Sie belegt klar eine durchschnittlich signifikant niedrigere Anzahl an Routingschritten. Die Ersparnis beträgt ca. einen Routingschritt, was auf die erhöhte Genauigkeit des Routingvorgangs zurückzuführen ist. Die Verteilungen der Pfadlänge bei aktivierter Chord Protokoll Erweiterung ist in Abbildung 5.2(b) dargestellt. Im Vergleich zum Standard Chord Protokoll sind sie spitzer und nach links verlagert. Dies ist auf die erhöhte Genauigkeit des Routing Vorgangs und die verringerte Anzahl an nötigen Routing Schritten zurückzuführen.

5.1.1.3 Routing des Chord Protokolls im dynamischen Netzwerk

Die Messungen in einem statischen Netzwerk sind idealisiert, denn dort sind alle Fingereinträge gemäß Chord Protokoll korrekt und alle Knoten des Netzwerks besitzen ihren korrekten Successor und Predecessor. In einem dynamischen Netzwerk ist dies nicht immer der Fall. Durch zum Netzwerk hinzukommende und vom Netzwerk abgehenden Knoten sind eventuell nicht alle Fingereinträge korrekt und nicht alle Knoten besitzen den laut Chord Protokoll vorgeschriebenen Successor und Predecessor. Entscheidend für die dynamischen Messungen ist hierbei das Verhältnis zwischen der Frequenz, mit der Veränderungen im Netzwerk auftreten, und der Häufigkeit, mit der die Successor- und Predecessor-Zuordnungen sowie die Einträge der Fingertabellen überprüft werden. Veränderungen werden umso schneller erkannt, je häufiger die Fingertabelleneinträge und Successor und Predecessor Zuordnungen überprüft und anschließend korrigiert werden. Allerdings erhöht sich der Verwaltungsaufwand des Chord Protokolls je häufiger die Knoten solche Überprüfungen vornehmen. Es ist deshalb immer eine



(a) Fehlerhafte Fingertabelleneinträge (b) Fehlerhafte Fingertabelleneinträge pro Knoten

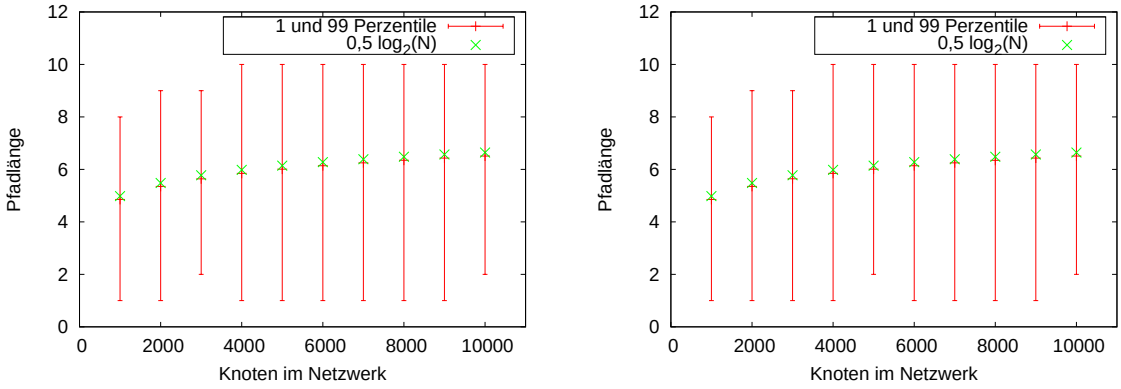
Abbildung 5.3: Fingereinträge im dynamischen Netzwerk; Chord Protokoll

Abwägung zwischen dem Aufwand und der zu erwartenden Änderungsfrequenz im Netzwerk zu treffen. Ein weiterer Faktor ist die Netzwerkgröße. Um zu entscheiden, welche Überprüfungsfrequenz für einen Peer im Netzwerk angemessen ist, ist es wichtig zu ermitteln, wie viele Änderungen im Netzwerk innerhalb einer Zeitspanne einen Peer durchschnittlich betreffen. Jeder Knoten besitzt 32 Fingereinträge. Entsprechend ist die Wahrscheinlichkeit, dass ein Fingereintrag aufgrund einer Änderung im Netzwerk korrigiert werden muss $32/N$, wobei N die Netzwerkgröße darstellt. Zusätzlich kann der Predecessor betroffen sein, was einer Wahrscheinlichkeit von $1/N$ entspricht. Zusammengefasst ist die Wahrscheinlichkeit, dass ein Knoten bei einer Änderung im Netzwerk betroffen ist $33/N$. Je größer das Netzwerk, desto kleiner ist die Wahrscheinlichkeit, dass ein Peer einen seiner Fingereinträge korrigieren muss. Diese drei Größen müssen für eine vergleichende Bewertung der Dynamik in einem Netzwerk betrachtet werden.

Für die Simulationen wird eine konstante Dynamik gewählt. Innerhalb einer Zeitspanne von 500 bzw. 1000 Simulationsschritten treten hierbei 50 Knoten dem Netzwerk bei, 30 Knoten verlassen das Netzwerk und 20 Knoten fallen aus. Dies entspricht einer Veränderung des Netzwerks alle fünf bzw. zehn Simulationsschritte. Die durch CHORD_STABILIZATION_STEPS und CHORD_FIX_FINGER_STEPS festgelegten Frequenzen zur Überprüfung der Successor und Predecessor Knoten sowie der Fingertabelleneinträgen bleiben hierbei auf ihren Standardeinstellungen (30 Simulationsschritte). Diese Dynamik wird für 10000 Simulationsschritte simuliert. Anschließend wird die Messung zur Ermittlung der Pfadlänge analog zu den statischen Messungen durchgeführt. Um eine Verfälschung der Ergebnisse zu verhindern, wird die Überprüfung und damit Korrektur der Successor- und Predecessor-Zuordnung sowie der Fingertabelleneinträge während der Messung der Pfadlänge deaktiviert.

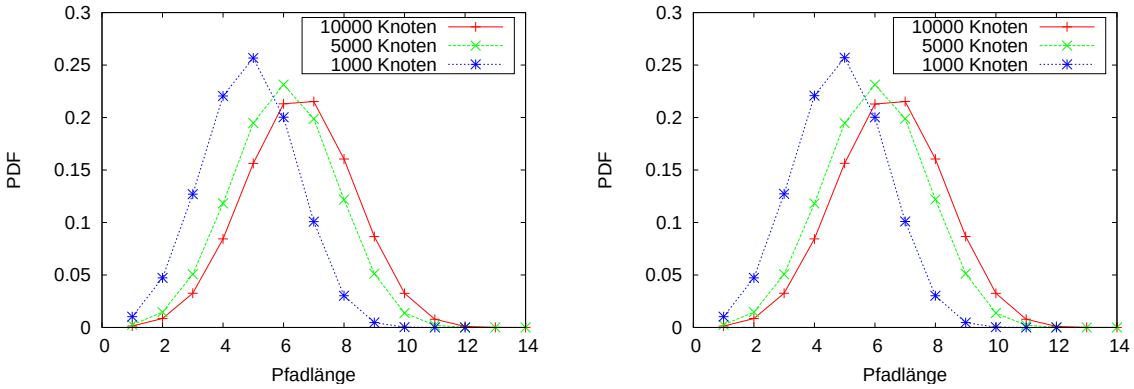
Entscheidend für die Ergebnisse ist der Zustand der Fingertabellen nach diesen 10000 Simulationsschritten. Dieser wird protokolliert und ist für die beschriebenen Netzwerkgrößen in Abbildung 5.3(a) dargestellt. Da die Anzahl der fehlerhaften Einträge relativ konstant bleibt, nimmt die Anzahl der fehlerhaften Einträge pro Knoten mit zunehmender Netzwerkgröße entsprechend ab. Dies ist in Abbildung 5.3(b) dargestellt.

Die Ergebnisse der Messungen der Pfadlänge im dynamischen Netzwerk sind in Abbildung 5.4(a) für die Änderungsfrequenz alle fünf Simulationsschritte und in Abbildung 5.4(b) mit der Änderungsfrequenz von zehn Simulationsschritten gezeigt. Wie dort zu sehen ist, hat die



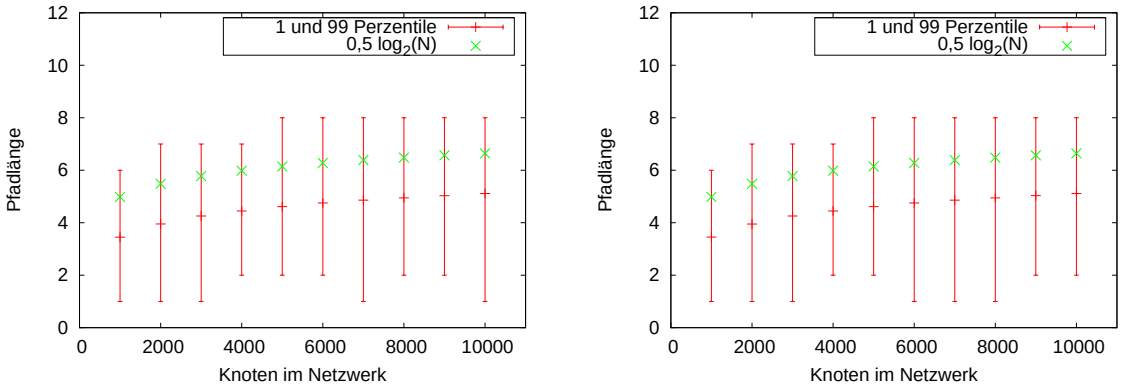
(a) Änderung alle fünf Simulationsschritte (b) Änderung alle zehn Simulationsschritte

Abbildung 5.4: Pfadlänge im dynamischen Netzwerk; Chord Protokoll



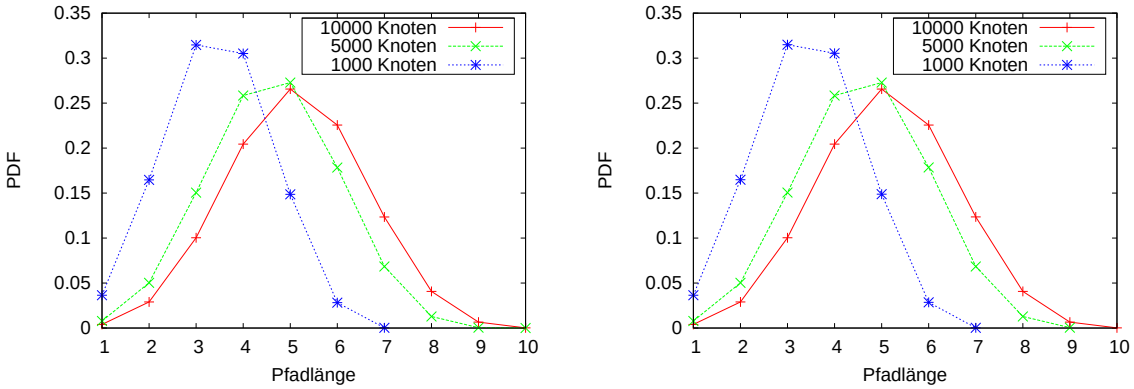
(a) Änderung alle fünf Simulationsschritte (b) Änderung alle zehn Simulationsschritte

Abbildung 5.5: Verteilung der Pfadlänge im dynamischen Netzwerk; Chord Protokoll



(a) Änderung alle fünf Simulationsschritte (b) Änderung alle zehn Simulationsschritte

Abbildung 5.6: Pfadlänge im dynamischen Netzwerk, erweitertes Chord Protokoll



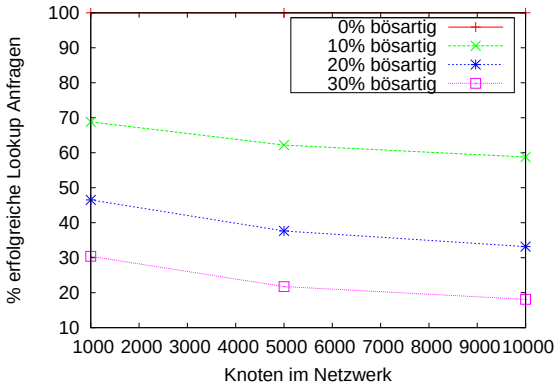
(a) Änderung alle fünf Simulationsschritte (b) Änderung alle zehn Simulationsschritte

Abbildung 5.7: Verteilung der Pfadlänge im dynamischen Netzwerk, erweitertes Chord

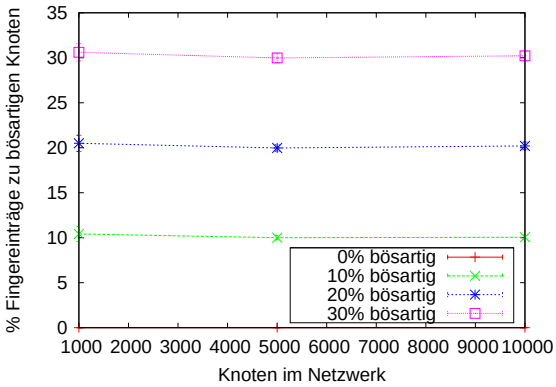
Änderungsfrequenz kaum Einfluss auf die Pfadlänge des Chord Netzwerks. Sie ist hier nur minimal größer als im statischen Fall. Selbiges gilt für die Verteilung, wie in Abbildung 5.5(a) für die Änderungsfrequenz fünf und in Abbildung 5.5(b) für die Änderungsfrequenz zehn zu sehen. Die Kurven sind auch hier nahezu identisch und im Vergleich zum statischen Netzwerk nur minimal nach rechts verschoben.

5.1.1.4 Routing erweitertes Chord Protokoll im dynamischen Netzwerk

Die für das Standard Chord Protokoll durchgeführten dynamischen Messungen werden für das erweiterte Chord Protokoll wiederholt, um auch für dieses die Auswirkungen der Dynamik zu zeigen. Die Ergebnisse für die Pfadlänge sind in Abbildung 5.6 zu sehen, die Verteilungen der Pfadlänge in Abbildung 5.7. Auch hier sind die Veränderungen gegenüber dem statischen Netzwerk mit maximalen Abweichungen bei der Pfadlänge von 0,002 vernachlässigbar. Der Zustand der Fingertabellen beim Messen der Pfadlänge entspricht bis auf minimale Abweichungen denen der Messungen des Standard Chord Protokolls. Eine Darstellung ihrer Zustände ist im Anhang B in Abbildung B.1 und B.2 dargestellt.



(a) Anteil erfolgreicher Anfragen im erweiterten Chord Netzwerk



(b) Fingereinträge zu bösartigen Knoten

Abbildung 5.8: Statisches Netzwerk mit bösartigen Knoten, erweitertes Chord

5.1.1.5 Erweitertes Chord Protokoll im statischen Netzwerk mit bösartigen Peers

Bei den bisherigen Messungen waren ausschließlich gutartige Peers im Chord Netzwerk vorhanden. Die Zuverlässigkeit des Chord Protokolls verschlechtert sich mit zunehmender Anzahl bösartiger Peers und dies selbst dann, wenn die Fingereinträge der Knoten im Netzwerk korrekt sind. Ursache hierfür ist, dass sich die bösartigen Knoten nicht am korrekten Ablauf des Protokolls beteiligen. Die wichtigsten Folgen sind:

- Datenobjekte, werden nicht mehr zuverlässig im Netzwerk gefunden deren Successor bösartig ist (d.h. bei der Abbildung des Datenobjekts in den Chord Schlüsselraum wird diesem eine ChordID zugeordnet, für die ein bösartiger Knoten verantwortlich ist).
- Wird eine Anfrage während des Routings zu einem bösartigen Peer weitergeleitet, ist deren korrekte Weiterleitung nicht mehr gewährleistet. Die Anfrage schlägt deshalb fehl.

Für die hier gezeigten Messungen wird die Erreichbarkeit und damit Zuverlässigkeit des Chord Netzwerks bei unterschiedlichen Anteilen von bösartigen Peers untersucht. Nach dem Aufbau des Netzwerks verhält sich ein festgelegter Anteil von zufällig ausgewählte Knoten bösartig.

Passive bösartige Peers Zunächst werden Netzwerke mit passiven bösartigen Peers betrachtet. *Bösartig* bedeutet in diesem Fall, dass sie die Nachrichten nicht mehr weiterleiten und das Routing einer Nachricht somit fehlschlägt. Wichtig ist, dass die bösartigen Knoten keinerlei Anstrengungen unternehmen, die Routinginformationen anderer Peers zu manipulieren (deshalb auch passiv). In Abbildung 5.8(a) sind die Ergebnisse für statische Netzwerke verschiedener Größen und unterschiedlichen Anteilen passiver bösartiger Peers gezeigt. Schon bei einem Anteil von nur 10% bösartigen Knoten liegt die Zuverlässigkeit bei unter 70%. Die Zuverlässigkeit verschlechtert sich mit zunehmender Netzwerkgröße, da sich die Anzahl der benötigten Routingschritte erhöht, was aus den vorherigen Ausführungen und Messungen hervorgeht (siehe Kapitel 5.1.1.1 und 5.1.1.2). Mit jedem zusätzlichen Routingschritt erhöht sich aber die Wahrscheinlichkeit eines Fehlschlags, denn bei jedem Routingschritt entspricht

die Wahrscheinlichkeit, dass die Nachricht an einen bösartigen Peer weitergeleitet wird, dem Anteil der bösartigen Knoten im Netzwerk. Für diese Messungen kommt das erweiterte Chord Protokoll zum Einsatz. Da es gegenüber dem Standard Chord Protokoll eine geringere Anzahl an Routing Schritten benötigt, ist für das Standard Chord Protokoll eine schlechtere Zuverlässigkeit zu erwarten. Diese Messungen zeigen, dass das Chord Protokoll nicht mehr funktionstüchtig ist, selbst wenn die bösartigen Peers darauf verzichten, die Fingertabelleneinträge der anderen Knoten zu manipulieren. Wie in Abbildung 5.8(b) zu sehen, entspricht deshalb der Anteil der Fingertabelleneinträge zu bösartigen Knoten dem Anteil dieser bösartigen Knoten im Netzwerk.

Bösartige Peers An dieser Stelle werden Netzwerke mit „normalen“ (aktiven) bösartigen Peers betrachtet. Bösartige Knoten versuchen, die Fingertabelleneinträge der anderen Knoten aktiv zu manipulieren. Ein Knoten n überprüft regelmäßig seine Fingertabelleneinträge, indem er nach dem $successor(n.finger[k].id)$ sucht. Hierzu werden FindSucc-Nachrichten mit den zu suchenden ChordIDs $n.finger[k].id$ ausgesendet und gemäß dem Chord Protokoll zum $successor(n.finger[k].id)$ geroutet. Bösartige Peers, die eine solche Nachricht erhalten, beantworten diese und geben sich so als $successor(n.finger[k].id)$ aus. Bösartig kooperierende Peers leiten die Nachricht an den $successor(n.finger[k].id)$ aus der Gruppe der kooperierenden bösartigen Peers weiter, der sich gegenüber dem anfragenden Peer als $successor(n.finger[k].id)$ ausgibt und die Anfrage beantwortet.

Durch dieses Verhalten gelingt es den bösartigen Peers, die Anzahl der Fingertabelleneinträge, die auf bösartige Knoten verweisen, über den Anteil der im Netzwerk vorhandenen bösartigen Knoten zu erhöhen. Dies ist eine der Hauptgefahren in strukturierten P2P-Netzwerken.

Um die Wirksamkeit dieser Angriffe zu verdeutlichen, wird ein zuvor intaktes statisches P2P-Netzwerk während fester Zeitabstände betrachtet. Als Maßstab für das Fortschreiten der Manipulation der Routinginformationen wird die Prozentzahl der falschen Einträge in den Fingertabellen der gutartigen Knoten angegeben. Diese wird regelmäßig während der Laufzeit der Simulation protokolliert und ist in Abbildung 5.9 für unterschiedliche Netzwerkgrößen mit variierendem Anteil bösartiger Peers dargestellt. Falsche Fingertabelleneinträge resultieren aus den Manipulationen der bösartigen Peers und verweisen deshalb zu bösartigen Knoten. Es handelt sich hier aber nicht um die Gesamtzahl der Verweise auf bösartige Knoten. Wie aus den vorhergehenden Messungen (siehe Abbildung 5.8(b)) hervorgeht, verweisen auch korrekte Fingereinträge auf bösartige Knoten. Hier wird deutlich, dass die Manipulation durch bösartige Knoten bereits ohne Veränderungen im Netzwerk sehr weitreichend ist. Selbst im besten Fall mit 1000 Knoten und 10% bösartiger Knoten sind innerhalb von 1000 Simulationsschritten schon 10% der Fingertabelleneinträge der gutartigen Peers falsch. Insgesamt verweisen somit in diesem Fall ca. 20% der Fingertabelleneinträge der gutartigen Knoten auf bösartige Knoten. Ausgehend von den 10% der Fingereinträge, die schon zuvor auf bösartige Knoten verwiesen haben, entspricht das einer Steigerung um 100%.

5.1.1.6 Erweitertes Chord Protokoll im dynamischen Netzwerk mit bösartigen Peers

Realistischere Werte als die Messungen im statischen Netzwerk liefert die Messung der Zuverlässigkeit des erweiterten Chord Protokolls in einem dynamischen Netzwerk. Der Simulationsaufbau entspricht hierbei dem zuvor vorgestellten Aufbau für dynamische Netzwerke; der

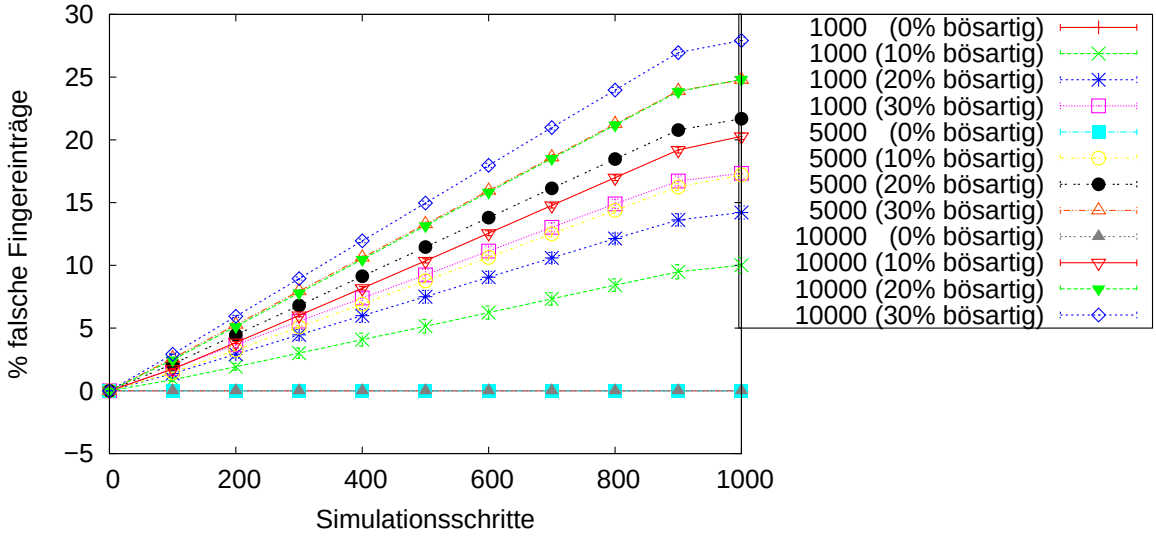


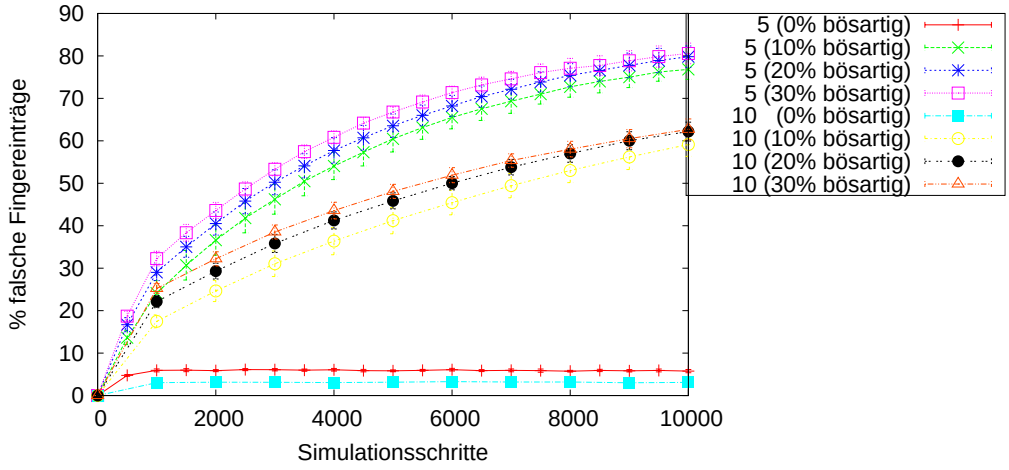
Abbildung 5.9: Fehlerhafter Fingereinträge im statischen Netzwerk

Anteil bösartiger Knoten wird nun variiert. Es handelt sich hierbei um „normale“ bösartige Peers, mit dem zuvor erläuterten Verhalten.

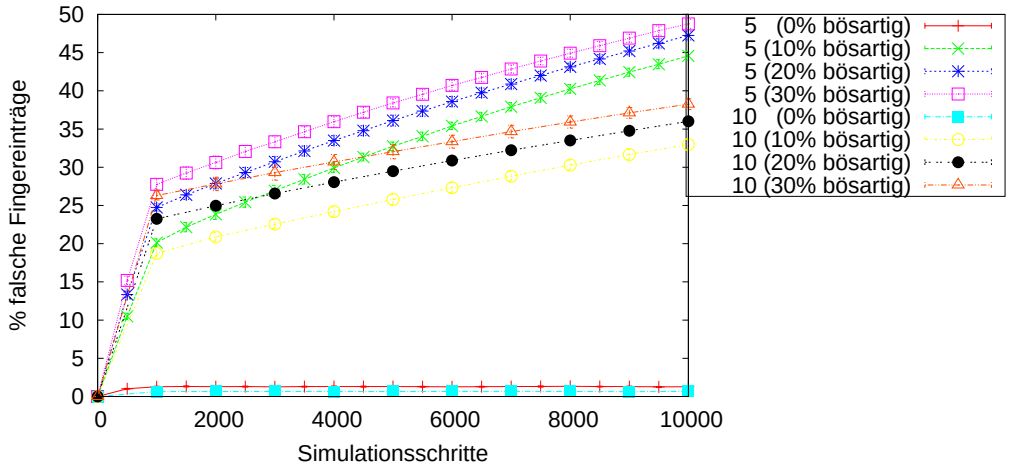
Verändert sich das Netzwerk dynamisch, erhalten bösartige Peers häufiger die Möglichkeit Manipulationen an den Fingertabellen vorzunehmen. Einträge der Fingertabellen bzw. Successor- oder Predecessor-Knoten verändern sich und müssen durch neue Knoten ersetzt werden, die über den vorgestellten Mechanismus gesucht werden.

Die Ergebnisse sind in Abbildung 5.10 für die Netzwerkgrößen 1000, 5000 und 10000 Knoten mit einer Änderungsfrequenz von fünf und zehn Simulationsschritten dargestellt. Die erste Zahl der Legende entspricht hierbei der Änderungsfrequenz. Wie in den Messungen zu sehen ist, verschlechtert sich der Zustand der Fingertabellen kontinuierlich mit zunehmender Simulationszeit. Zunächst gilt, dass der Zustand der Fingertabellen umso schlechter wird, je höher die prozentuale Fluktuation im Netzwerk ist. Da die Simulationszeit bei allen Messungen gleich ist, gibt es 2000 Änderungen bei Änderungsfrequenz fünf bzw. 1000 Änderungen bei Änderungsfrequenz zehn. Hierdurch ist die prozentuale Fluktuation im 1000 Knoten umfassenden Netzwerk zehnmal höher als im Netzwerk mit 10000 Knoten. Entsprechend sind die Ergebnisse umso schlechter, je kleiner das Netzwerk ist. Zudem ist bei allen Messungen zu erkennen, dass bei der Änderungsfrequenz von zehn die Fingertabellen in einem besseren Zustand sind. Der prozentuale Anteil an bösartigen Peers im Netzwerk hat hingegen nur einen kleinen Einfluss auf den Zustand der Fingertabellen. Dies zeigt, wie erfolgreich die Manipulation der Routingtabellen durch die bösartigen Peers ist, geht doch nur ein geringer Anteil der Fehler auf die Fluktuation im Netzwerk zurück. Dies ist gut an den Messergebnissen der Netzwerke ohne bösartige Peers zu erkennen. Deren Anteil fehlerhafter Fingereinträge pendelt sich in Abhängigkeit von der Änderungsfrequenz und Netzwerkgröße bei 5% (1000 Knoten, Änderungsfrequenz fünf) oder weniger ein.

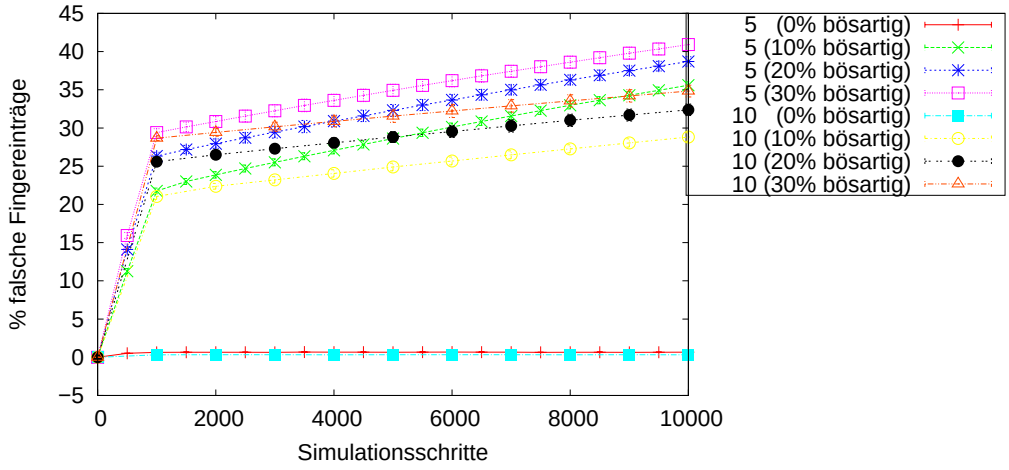
Der charakteristische Knick in den Kurven der Abbildungen 5.10, ist durch den Versuchsaufbau bedingt. In der Startphase geschieht die Manipulation der Fingertabellen bei der autonomen regelmäßigen Überprüfung der Fingertabelleneinträge durch die einzelnen Peers (wie zuvor in Kapitel 5.1.1.5 dargestellt). Beim erstmaligen Überprüfen eines Fingertabellen-



(a) 1000 Knoten mit Änderungsfrequenz fünf und zehn



(b) 5000 Knoten mit Änderungsfrequenz fünf und zehn



(c) 10000 Knoten mit Änderungsfrequenz fünf und zehn

Abbildung 5.10: Prozentzahl fehlerhafter Fingereinträge im dynamischen Netzwerk

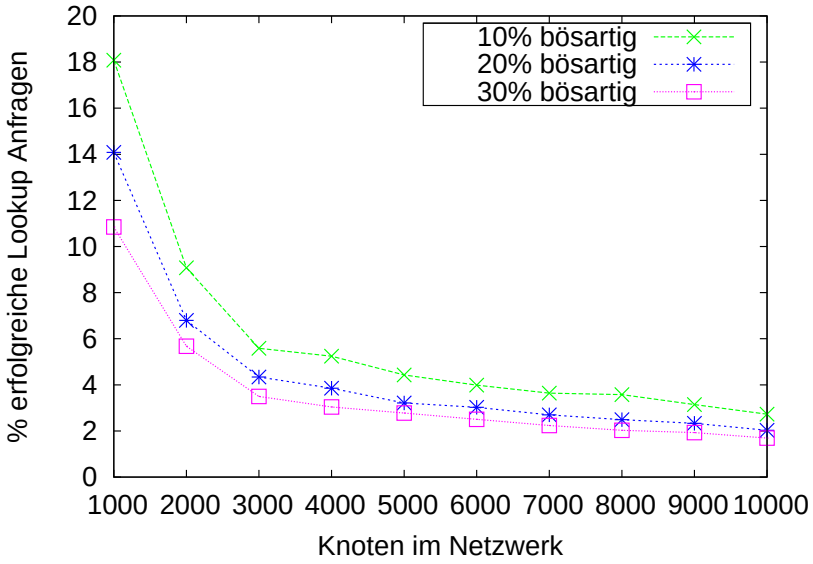


Abbildung 5.11: Zuverlässigkeit im dynamischen Chord Netzwerk (Freq 10)

eintrags mit den nun neu vorhandenen bösartigen Peers im Netzwerk, ist die Erfolgswahrscheinlichkeit einer Manipulation besonders hoch. Danach sinkt sie merklich ab und wird durch andere Effekte wie Fluktuation und dem Zustand der Fingertabellen anderer Peers beeinflusst. Der Knick in den Kurven beschreibt deshalb exakt zu dem Zeitpunkt, an dem alle Fingereinträge der Peers einmalig überprüft worden sind.

Die hier gezeigte Qualität der Fingertabellen schließt einen zuverlässigen Ablauf des Chord Protokolls aus. Das Netzwerk zerbricht schon nach kürzester Zeit in Teilnetzwerke. Neu hinzukommende Peers gliedern sich in eines der Teilnetzwerke ein. Dies belegt eindrücklich eine Messung der Erreichbarkeit der anderen Knoten im Netzwerk. Die Ergebnisse für diese Messungen nach 10000 Simulationsschritten mit einer Änderung im Netzwerk alle zehn Simulationsschritte sind in Abbildung 5.11 zu sehen. Wie schon anhand des Zustands der Fingertabelle zu erwarten war, funktioniert das Chord Protokoll schon bei kleinen Netzwerken nicht mehr. In allen dort dargestellten Fällen liegt die Erreichbarkeit unterhalb 20% und nähert sich mit zunehmender Netzwerkgröße schnell der 2% Marke an.

Die Ausführungen belegen die Notwendigkeit der in Kapitel 3.5.7 dargelegten Erweiterungen des etablierten Chord Protokolls. Deren Wirksamkeit wird im Folgenden mit Messungen belegt.

5.1.2 Verlässliches Chord Protokoll

Das Chord Protokoll wird um zwei wesentliche Techniken erweitert: Den Fehlertest, mit dem fehlerhafte Antworten erkannt werden können und die verlässliche Weiterleitung von Nachrichten durch das redundante Routing (siehe Kapitel 3.5.7.3).

5.1.2.1 Redundantes Routing

Wichtig für das redundante Routing ist, wie schon in den Ausführungen in Kapitel 3.5.7.3 dargelegt, dass in der ersten Phase des redundanten Routing Algorithmus versendete FindNeigh-

bour-Nachrichten einen gutartigen Nachbarn der gesuchten ChordID finden. Hierzu muss eine ausreichend große Anzahl an redundanten Routingpfaden beschritten werden. Bei dieser Betrachtung wird davon ausgegangen, dass die Fingereinträge durch bösartige Peers nicht manipuliert werden können. Dies kann durch entsprechende Vorkehrungen im verlässlichen Chord Protokoll sichergestellt werden. Die schon beschriebenen Maßnahmen sind hierbei:

- Bedingungen für Fingertabelleneinträge
- Einsatz von redundantem Routing zur Überprüfung der Fingertabelleneinträge
- Zufällige Festlegung der ChordID
- Auswahl der Bootstrap-Knoten über die Zertifizierungsinstanz

Unter diesen Voraussetzungen lässt sich die Anzahl nötiger redundanter Routingpfade statistisch berechnen. Diese Berechnungen sind im Anhang D dargestellt. Daraus geht hervor, dass z.B. für eine Netzwerkgröße von 10000 Knoten 100 verschiedene Pfade beschritten werden müssen. Dies gelingt, indem msg_hop_1 und msg_hop_2 auf 12 gesetzt werden.

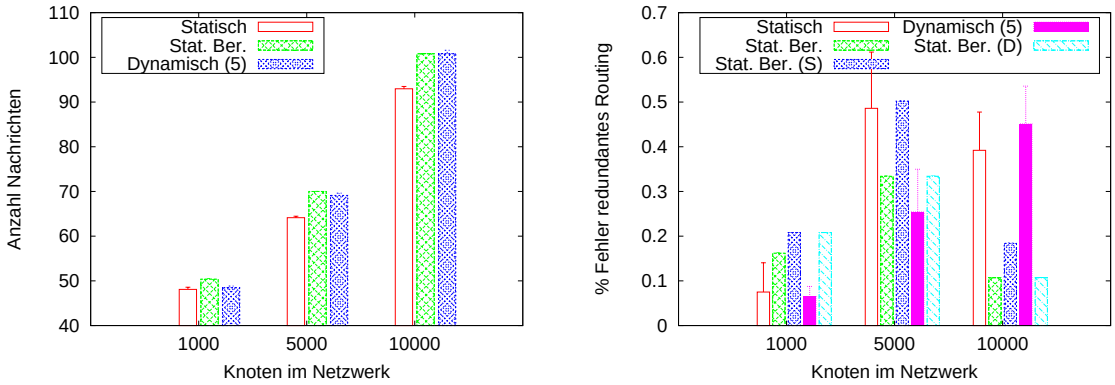
Die so statistisch berechneten Vorgabewerte werden durch eine Simulation überprüft. Für den Simulationsaufbau wird ein Netzwerk erzeugt, das ausschließlich aus gutartigen Knoten besteht. Hierbei kommt das beschriebene erweiterte Chord Protokoll zum Einsatz. Anschließend ändern 30% zufällig ausgewählte Knoten ihr Verhalten zu bösartig kooperierend.

Bösartig kooperierendes Verhalten bedeutet hier, dass die FindNeighbour-Nachrichten des redundanten Routingalgorithmus nicht beantwortet oder weitergeleitet werden. Bösartig kooperierende Peers unterbrechen damit die weitere Verbreitung der Nachrichten und reduzieren so die Chance, dass Nachbarn für die angefragte ChordID gefunden werden. Es ist erklärtes Ziel der bösartig kooperierenden Peers zu verhindern, dass eine sichere Knoten Menge (SKM) erzeugt werden kann und damit Manipulationen erkannt werden. NodeList- und Forward-Nachrichten werden von den bösartig kooperierenden Knoten beantwortet. Bei NodeList-Nachrichten liefern sie allerdings nur andere bösartig kooperierende Knoten zurück. Da im Stadium des Protokolls, in dem diese Nachrichten erzeugt werden, das erfolgreiche Erzeugen einer SKM nicht mehr verhindert werden kann, versuchen sie nun, möglichst viele andere bösartige Knoten in die SKM zu schleusen und so deren Qualität zu mindern.

Bei der Simulation suchen zufällig ausgewählte gutartige Knoten für 10000 zufällig ausgewählte ChordIDs $SKMs$. Zudem wird die Simulationen zehnmal mit unterschiedlicher „Seed“ des Zufallszahlengenerators gestartet. Die dargestellten Ergebnisse entsprechen den ermittelten Durchschnittswerten.

Kritisch bei der Ermittlung einer SKM ist, ob eine Nachricht einen gutartigen Knoten erreicht, in dessen Replikationsgruppe sich die gesuchte ChordID befindet. Dies wird durch das redundante Versenden der Nachrichten sichergestellt. Entsprechend ist ein Erhalt einer SKM für eine Anfrage gleichbedeutend mit dem Erreichen eines gutartigen Knotens. Die Replikationsgruppe hat in PACS die Größe 32.

In Abbildung 5.12(a) wird die statistische Abschätzung der Anzahl der Nachrichten mit der in der Simulation gemessenen Anzahl verglichen. Neben den hier als „statisch“ bezeichneten Messergebnissen der statischen Simulation sind auch die Ergebnisse der dynamischen Simulation dargestellt, die als „dynamisch (5)“ gekennzeichnet sind und später erläutert werden. Hierbei ist zu beachten, dass mit unterschiedlichen msg_hop_1 und msg_hop_2 gearbeitet wird. Für 1000 Knoten ist $msg\_hop\_1 = 12$ und $msg\_hop\_2 = 6$, bei 5000 Knoten ist $msg\_hop\_1 = 10$ und $msg\_hop\_2 = 10$ und bei 10000 Knoten ist $msg\_hop\_1 = 12$ und



(a) Anzahl FindNeighbour-Nachrichten

(b) Zuverlässigkeit redundantes Routing

Abbildung 5.12: Redundantes Routing bei verschiedenen Netzwerkgrößen

$msg\_hop\_2 = 12$. Die als statistische Berechnung „stat. Ber“ bezeichneten Balken entsprechen den Berechnungen im Anhang D.

Die Anzahl der Nachrichten ist bei der statischen Simulation zwischen zwischen 2,3 (1000 Knoten) und 7,8 (10000 Knoten) Nachrichten kleiner als statistisch vorausgesagt. Grund hierfür ist, dass bei einigen Knoten mehrere Einträge der Fingertabelle auf den gleichen Peer verweisen. Deshalb kann nicht die erforderliche Anzahl an redundanten Nachrichten erzeugt werden, was zu der beobachteten Abweichung führt.

Die Ergebnisse der Simulation bezüglich der Zuverlässigkeit im Vergleich mit den Ergebnissen der statistischen Berechnung sind in Abbildung 5.12(b) zu sehen. Die korrigierten statistischen Berechnungen „stat. Ber.(S)“ berücksichtigen die bei der Simulation durch das redundante Routing tatsächlich erzeugten Mengen an Nachrichten und berechnen aus diesen die Fehlerwahrscheinlichkeit. Wie in Abbildung 5.12(a) zu sehen, unterscheiden sie sich aber nur minimal. Die Abweichungen der Messungen von den Berechnungen liegen bei maximal 0,4%. Interessant ist hierbei ihre positive bzw. negative Auswirkung. Bei 1000 und 5000 Knoten im Netzwerk sind die Ergebnisse besser als die statistisch berechneten. Dies liegt an der bei der Berechnung nicht berücksichtigten Chord Routing Erweiterung, die für PACS entwickelt wurde. Bei 10000 Knoten ist die Zuverlässigkeit schlechter als statistisch berechnet. Dies resultiert aus der vereinfachten Annahme bei der statistischen Berechnung und der Chord Routing Erweiterung, die aufgrund des größeren Netzwerks keinen großen Einfluss mehr hat.

Insgesamt sind die Unterschiede zwischen statistischer Berechnung und den Messergebnissen sehr gering. Deshalb wird auf weitergehende Messungen mit verschiedenen Varianten verzichtet.

Sobald Knoten das Netzwerk verlassen und neue Knoten hinzukommen, müssen die Knoten ihre Fingertabelleneinträge sowie Successor- und Predecessor-Knoten regelmäßig überprüfen und neuen Gegebenheiten anpassen. Um Manipulationen durch bösartige Peers zu verhindern, ist hierzu die sichere Verwaltung der Fingereinträge erforderlich. Die Ergebnisse eines dynamischen Netzwerks sind deshalb nicht mehr dezidiert auf den redundanten Routing Algorithmus zurückzuführen. Die in Abbildung 5.12 als „Dynamisch (5)“ bezeichneten Ergebnisse werden wie folgt ermittelt. Direkt beim sukzessiven Aufbau des Netzwerks ist ein Anteil von 30% bösartiger Knoten vorhanden. Die Mechanismen zur sicheren Verwaltung der Routinginformationen stellen hierbei sicher, dass trotz des Vorhandenseins dieser bösartig

kooperierenden Peers das Netzwerk nach wie vor intakt bleibt. Sobald das Netzwerk in seiner festgelegten Größe aufgebaut ist, beantragen zufällig ausgewählte gutartige Knoten $SKMs$ für 10000 zufällig ausgewählte ChordIDs. Hierbei wird darauf geachtet, dass Knoten, die gerade eine SKM beantragt haben, das Netzwerk nicht verlassen, während sie auf das Ergebnis warten. Änderungen am Netzwerk treten hierbei alle fünf Simulationsschritte auf.

Wie in Abbildung 5.12(a) zu sehen, werden im dynamischen Netzwerk mehr Nachrichten erzeugt als im statischen Netzwerk. Grund hierfür ist, dass die Knoten in ihren Fingertabellen z.T. noch Verweise zu schon nicht mehr im Netzwerk befindlichen Knoten enthalten. Dadurch verschicken sie mehr Nachrichten. Dieses Mehr an Nachrichten trägt aber nicht zur Steigerung der Zuverlässigkeit bei. Die Werte der korrigierten statistischen Berechnungen „stat. Ber.(D)“ für die dynamische Simulation sind aufgrund der höheren Anzahl der Nachrichten besser, als die der statischen. Trotz der Dynamik im Netzwerk, und der damit nicht immer korrekten Fingertabellen der Knoten, sind die Resultate des redundanten Routing Algorithmus nur leicht schlechter und bei 5000 Knoten sogar leicht besser. Da die sehr geringen Unterschiede zwischen den gemessenen Werten fast innerhalb der Standardabweichung der hier dargestellten aggregierten Messergebnisse liegen, kann eine wertende Aussage hier nicht vorgenommen werden kann.

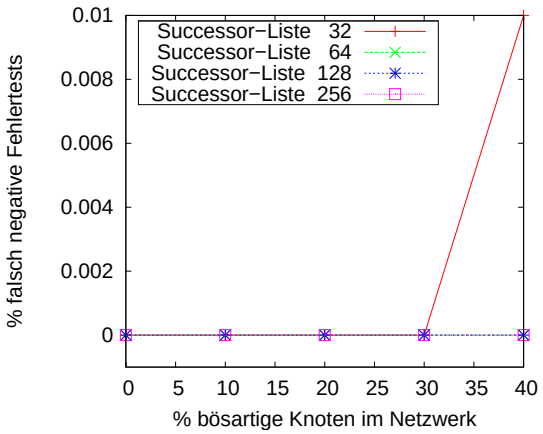
Zusammenfassend gilt, dass die Messergebnisse für die statische und auch die dynamische Simulation im Wesentlichen den statistisch berechneten Werten entsprechen. Bei allen Simulationen liegt die Zuverlässigkeit des redundanten Routing Algorithmus oberhalb 99,5%. Damit ist die Funktionstüchtigkeit dieses Algorithmus belegt. Mit ihm lassen sich alle Knoten des Netzwerks selbst bei 30% bösartig kooperierenden Peers im Netzwerk zuverlässig finden.

Da eine Änderung alle fünf Simulationsschritte die hier betrachtete maximale Dynamik im Netzwerk darstellt, wird auf eine Darstellung von Ergebnissen mit geringerer Dynamik verzichtet. Die Ergebnisse nähern sich mit abnehmender Dynamik denen der statischen Messung an.

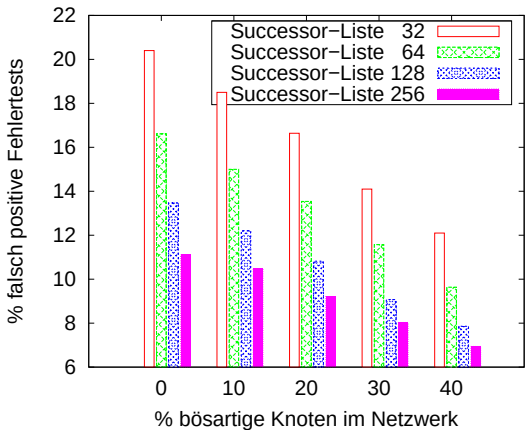
5.1.2.2 Zuverlässigkeit des Fehlertests

Die Zuverlässigkeit des Fehlertests wird anhand weiterer Messungen belegt. Der Wert für γ wird hierbei auf den nach Berechnungen von Castro u.a. [CDG+02] optimalen Wert $\gamma = 1{,}23$ festgesetzt. Um die Zuverlässigkeit zu prüfen, kommt zunächst wieder ein statisches Chord Netzwerk mit 10000 Knoten mit unterschiedlichem Anteil bösartiger Peers zum Einsatz. Hier wird darauf geachtet, dass sich das Netzwerk stabilisiert hat, d.h. alle Knoten besitzen den nach dem Protokoll vorgeschriebenen korrekten Successor und Predecessor. Auch die Fingereinträge verweisen auf die nach dem Protokoll festgelegten Knoten $successor(n.finger[k].id)$. Die Zuverlässigkeit des Fehlertests wird überprüft, indem zufällig ausgewählte gutartige Knoten von anderen zufällig ausgewählten Knoten deren AKM anfordern. Gute Knoten liefern hierbei immer eine korrekte Antwort Knoten Menge (AKM), böse Knoten immer eine AKM mit ausschließlich bösartigen Knoten an den anfragenden Peer zurück. Für die zurückgelieferte AKM nimmt der Knoten den Fehlertest vor und protokolliert das Ergebnis. Für die hier dargestellten Ergebnisse werden 10000 solcher Überprüfungen vorgenommen. Zudem werden die Simulationen zehnmal mit unterschiedlicher „Seed“ des Zufallszahlengenerators gestartet und der Durchschnitt aller Ergebnisse berechnet.

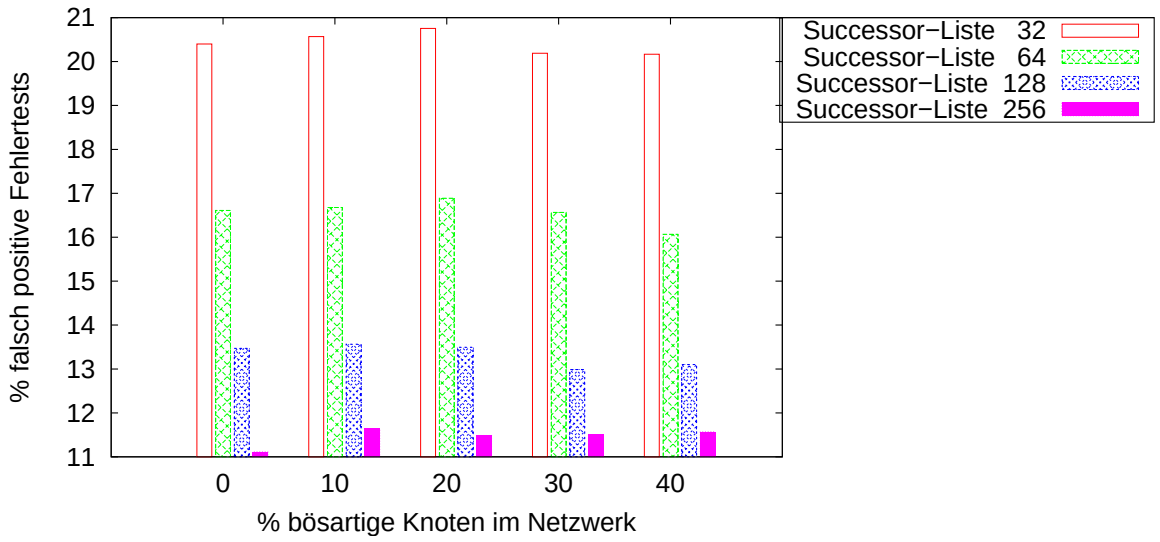
Für die in Abbildung 5.13(a) gezeigten Messungen wird die Größe der Successor-Liste variiert. Wie bereits bei den Ausführungen zum Fehlertest erläutert, erhöht sich die Genauigkeit der Tests mit der Größe der Successor-Liste. Selbst bei einer Successor-Listen Größe von 32



(a) Falsch negative Fehlertests



(b) Falsch positive Fehlertests

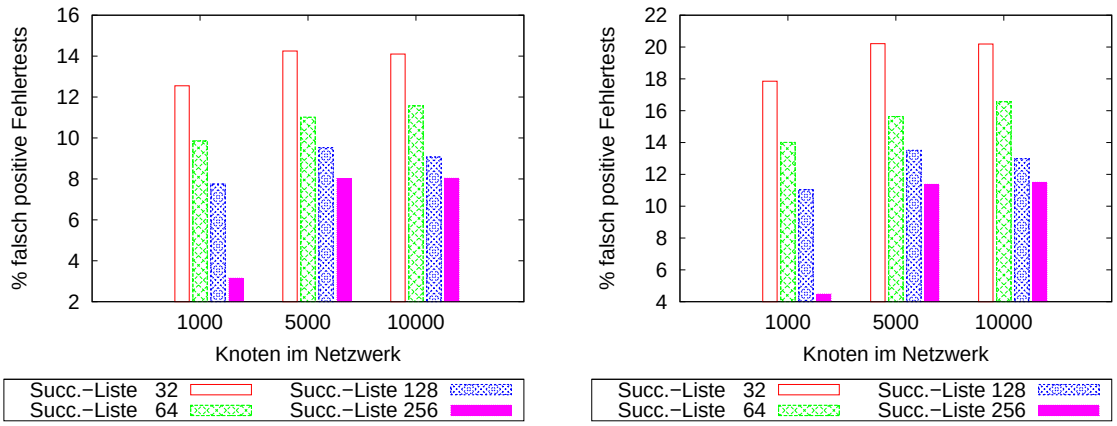


(c) Relativer Anteil falsch positiver Fehlertests

Abbildung 5.13: Zuverlässigkeit Fehlertest bei verschiedenen Successor-Listen-Größen

werden die bösartigen $AKMs$ zuverlässig erkannt. Innerhalb des spezifizierten Operationsrahmens geschieht dies sogar ohne jegliche Fehler. D.h. Knotenmengen, die ausschließlich aus bösartigen Peers bestehen, werden garantiert erkannt. Diese Zuverlässigkeit hat allerdings ihren Preis. So gibt es viele falsch positive Testresultate. Dies ist in Abbildung 5.13(b) wiederum für verschiedene Successor-Listen-Größen dargestellt.

Hier wird auch die größere Genauigkeit des Fehlertests bei einer größeren Successor-Liste deutlich, denn der Anteil falsch positiver Resultate ist umso kleiner, je größer die Successor-Liste ist (bei gleichbleibendem Anteil bösartiger Peers). So sind bei einer Successor-Listen-Größe von 256 lediglich ca. 12% falsch positive Resultate zu verzeichnen. Gut zu beobachten ist außerdem, dass sich deren Anzahl mit zunehmenden Anteil bösartiger Knoten verringert. Grund hierfür ist der geringere Anteil gutartiger $AKMs$ an der Gesamtmenge. Entsprechend reduziert sich auch der Anteil der falsch positiven Resultate. Dies ist in Abbildung 5.13(c)



(a) Falsch positive Fehlertests

(b) Positive Fehlertests bei korrekten $AKMs$

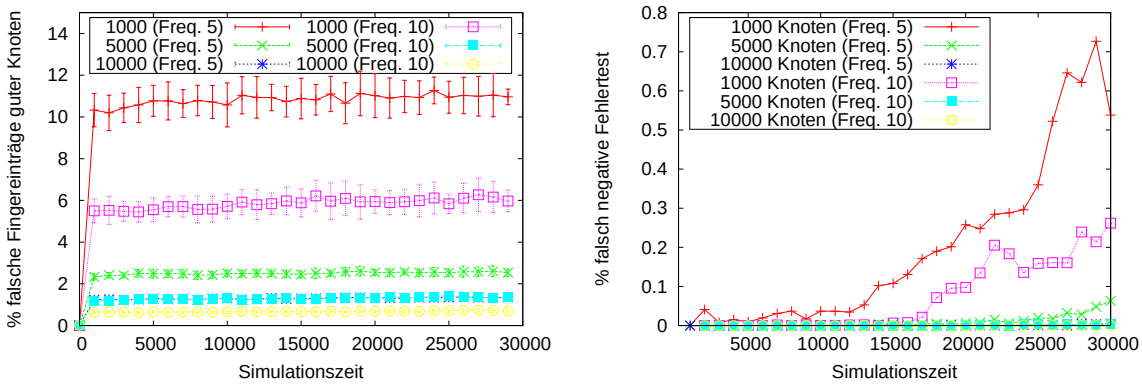
Abbildung 5.14: Zuverlässigkeit Fehlertest bei 30% bösartigen Knoten

ersichtlich, in der die relativen Prozentzahlen zu allen gutartigen Knotenmengen gezeigt werden.

Die Größe des Netzwerks hat Einfluss auf die Genauigkeit des Fehlertests. Basis des Fehlertests ist die Successor-Liste des Peers, der die Überprüfung vornimmt, sowie die vom antwortenden Peer zurückgelieferte AKM. Je größer der durch die Successor-Liste abgedeckte Teil des Netzwerks ist, desto exakter arbeitet der Fehlertest. Dies ist in Abbildung 5.14(a) absolut und in Abbildung 5.14(b) relativ dargestellt. Hierbei wird der Anteil bösartiger Peers konstant bei 30% gehalten, während die Netzwerkgröße variiert. Wie ersichtlich, sind die Schwankungen bei einer Successor-Liste der Größe 256 mit bis zu 7% am größten. Auffällig ist vor allem der sehr niedrige Wert im 1000 Knoten Netzwerk. Dies erklärt sich dadurch, dass die Successor-Liste ein Viertel des Netzwerks abdeckt und deshalb entsprechend genau arbeitet. Bei einer kleineren Successor-Liste ist dieser Anteil entsprechend geringer und so verschlechtern sich die Ergebnisse des Fehlertests weniger deutlich. Die Fehlertestergebnisse für eine Successor-Liste mit Größe 128 und 256 unterscheiden sich nur minimal zwischen den Netzwerkgrößen 5000 und 10000. Es ist deshalb davon auszugehen, dass eine Netzwerkgröße ab 5000 Knoten keinen Einfluss mehr auf den Fehlertest mit diesen Successor-Listen-Größen besitzt.

Wichtigste Erkenntnis ist, dass die Zuverlässigkeit des Fehlertests nicht von der Netzwerkgröße abhängt. Bei 30% bösartigen Knoten treten bei allen hier dargestellten Netzwerkgrößen keine falsch negativen Fehlertests auf. Auf eine Darstellung in einem Diagramm wird deshalb verzichtet. Die Darstellungen der Ergebnisse für 0%, 10%, 20% und 40% bösartige Knoten im Netzwerk sind im Anhang C dargestellt.

Wie zuvor stellt das statische Netzwerk eine idealtypische Umgebung dar. In einem dynamischen Netzwerk ist deshalb zu erwarten, dass die Ergebnisse schlechter ausfallen, da durch die Veränderungen im Netzwerk die Fingertabellen sowie die Successor- und Predecessor-Listen veraltete Einträge enthalten. Die Messungen finden deshalb nach einer vorher festgelegten Anzahl an Simulationsschritten und damit Änderungen im Netzwerk statt. Hierbei werden neben den Ergebnissen des Fehlertests auch der Zustand der Fingertabellen sowie Successor- und Predecessor-Listen protokolliert. Der zeitliche Verlauf des Zustands der Fingertabellen ist in Abbildung 5.15(a) zu sehen. In dieser Abbildung werden nur die Fehler in den Fingertabel-



(a) Zustand der Fingertabellen

(b) Falsch negative Fehlertests

Abbildung 5.15: Fehlertestzuverlässigkeit im dynamischen Netzwerk (30% bösartig)

len der gutartigen Knoten aufgeführt. Die Einträge in den Fingertabellen bösartiger Knoten sind aufgrund ihres bösartigen Verhaltens ohnehin beliebig falsch. Ein Fehler ist hierbei ein aufgrund der Änderungen im Netzwerk nicht mehr korrekter Eintrag in der Fingertabelle, d.h. $n.finger[k].knoten \neq successor(n.finger[k].id)$, sowie alle Fingereinträge, die leer sind. Leere Fingertabelleneinträge können durch das Verschwinden von Knoten entstehen. Stellt ein Knoten die Abwesenheit eines Fingerknotens fest, löscht er diesen zunächst nur. Die leeren Fingereinträge werden erst bei der routinemäßigen Überprüfung des Eintrags durch den dann aktuellen Eintrag ersetzt.

Wie in Abbildung 5.15(a) ersichtlich, verschlechtert sich der Zustand der Fingertabellen zunächst sprunghaft, um sich dann auf ein Niveau einzupendeln, das von Netzwerkgröße und Änderungsfrequenz im Netzwerk abhängt. Bei einer CHORD_STABILIZATION_STEPS Anzahl von konstant 30 wird deutlich, dass dieses Niveau mit der Änderungsfrequenz steigt. Zudem ist zu erkennen, dass kleine Netzwerke bei gleichbleibender Änderungsfrequenz einen größeren Anteil falscher Fingertabelleneinträge besitzen als größere. Dies erklärt sich daraus, dass dort ein Knoten wesentlich häufiger Änderungen unterzogen ist als in einem größeren Netzwerk.

Aufgrund der hier gezeigten Entwicklung ist es wenig verwunderlich, dass auch die Zuverlässigkeit des Fehlertests mit der Änderungsfrequenz variiert. Diese ist in Abbildung 5.15(b) zu sehen. Über die Simulationszeit wird die Zuverlässigkeit des Fehlertests bei unterschiedlichen Netzwerkgrößen und Änderungsfrequenzen aufgezeichnet. Wie zu sehen, ist lediglich bei kleinen Netzwerken mit 1000 Knoten mit fortschreitender Simulationszeit ein Anstieg der falsch negativen Fehlertests auf bis zu 0,75% zu verzeichnen. Bei größeren Netzwerken sind die Auswirkungen mit 0,05% bei 5000 Knoten und 0,005% bei 10000 Knoten nicht annähernd so deutlich. Generell sind die auftretenden Verschlechterungen bei einer Änderungsfrequenz von fünf Simulationsschritten größer als bei zehn Simulationsschritten.

Zusammenfassend wird die Zuverlässigkeit des Fehlertests durch fehlerhafte Einträge in den Fingertabellen beeinträchtigt, die mit Fehlern in den Successor- und Predecessor-Listen korrelieren. Jedoch sind die Beeinträchtigungen nur gering und selbst im schlechtesten Fall unterhalb der 1% Prozent Schwelle, weshalb der Fehlertest selbst dort als zuverlässig bezeichnet werden kann.

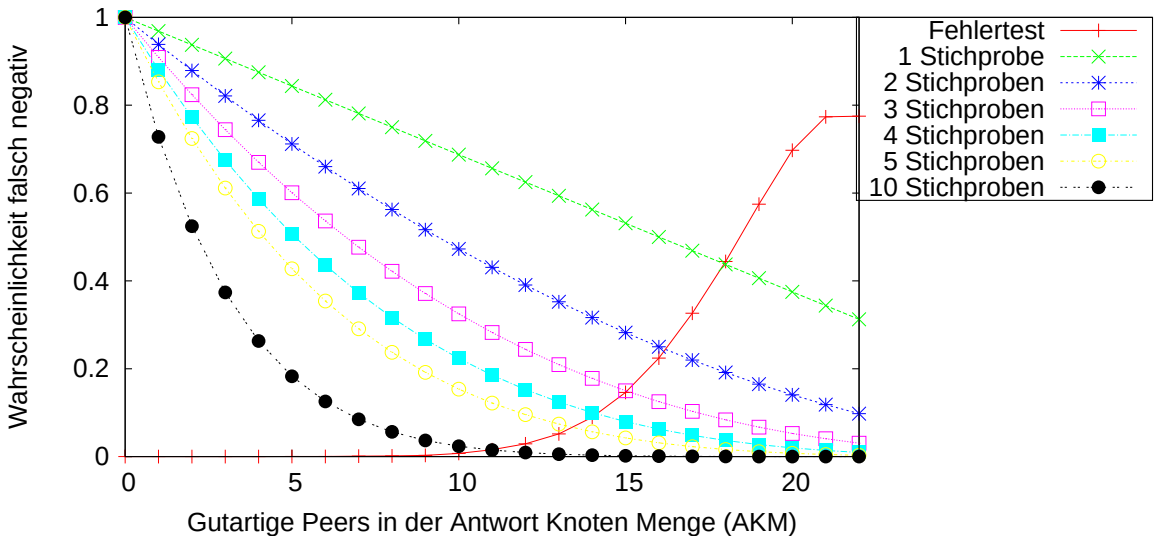


Abbildung 5.16: Nichterkennung falscher Antworten bei gemischten $AKMs$

Kombination Fehlertest und Stichprobentest Wie in Kapitel 3.5.7.3 erläutert, wird bislang davon ausgegangen, dass bösartige Peers eine AKM zurückliefern, die ausschließlich aus bösartigen Peers besteht. Dies setzt voraus, dass der anfragende Peer auch alle Knoten der AKM nochmals anfragt, was bei einer AKM-Größe von 32 Knoten sehr aufwändig ist. Um diesen Aufwand zu reduzieren, werden in PACS nicht alle Knoten kontrolliert, sondern nur eine Stichprobe genommen. Bösartige Knoten haben so die Möglichkeit, auch gutartige Knoten in die AKM zu mischen und darauf zu vertrauen, dass nicht alle Knoten überprüft werden. Die Wahrscheinlichkeit einen gutartigen Peer bei einer Stichprobe auszuwählen steigt hierbei mit der Anzahl der gutartigen Peers in der AKM und der Anzahl der Stichproben. Dies ist in Abbildung 5.16 klar abzulesen. Der Fehlertest arbeitet mit zunehmender Anzahl gutartiger Knoten nicht mehr zuverlässig genug. Bei den für diese Messung gewählten Einstellungen (AKM der Größe 32, $\gamma = 1,23$, Successor-Liste der Größe 256, statisches Netzwerk mit 10000 Knoten) erkennt der Fehlertest die bösartigen $AKMs$ bei bis zu zehn gutartigen Knoten in der bösartigen Knotenmenge noch zuverlässig. Ab zehn gutartigen Knoten verschlechtert sich das Ergebnis zunehmend. Ab 22 gutartigen Knoten in der AKM handelt es sich um eine AKM, wie sie auch von gutartigen Peers zurückgeliefert wird. Ausgehend von 30% bösartigen Knoten im Netzwerk beinhalten die Replikationsgruppe und die Predecessor-Liste ebenfalls 30% bösartige Knoten. Bei der Größe der AKM von 32 sind dies ca. zehn bösartige Knoten. Der Stichprobentest hingegen arbeitet umso zuverlässiger, je größer die Stichprobe ist und je mehr gutartige Peers sich in der AKM befinden. Da beide Tests unabhängig voneinander sind, liegt die Lösung in einer Kombination des Stichprobentests mit dem Fehlertest. Hieraus ergibt sich die in Abbildung 5.17 dargestellte Wahrscheinlichkeit für falsch negative Testresultate. Da bei der Kombination von Fehlertest und lediglich fünf Stichproben die Wahrscheinlichkeit von falsch negativen Testergebnissen zuverlässig unter 1% liegt, wird diese Größe für PACS-interne Verwaltungsnachrichten festgelegt.

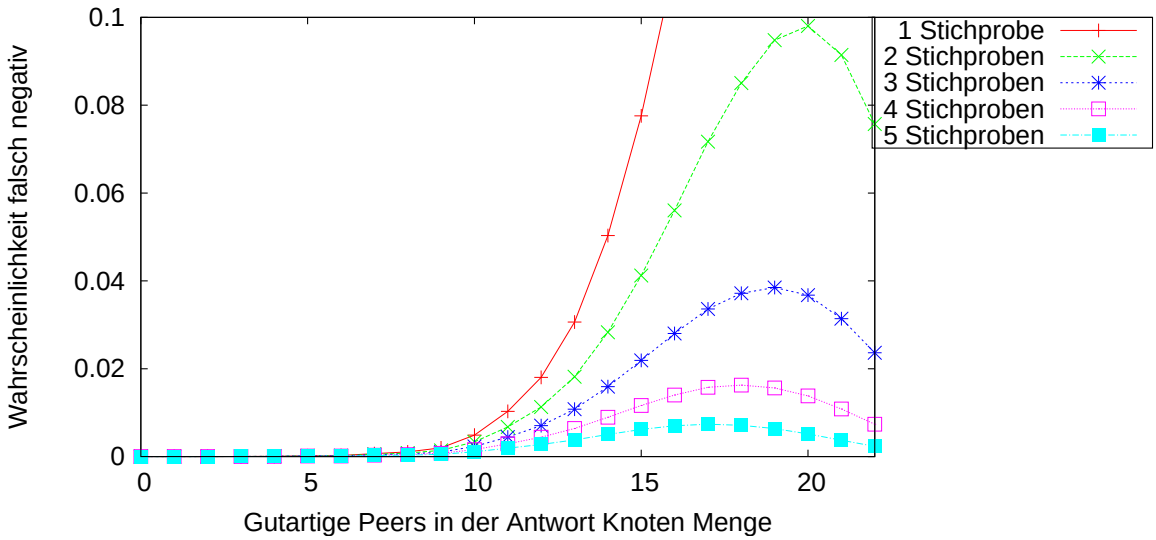


Abbildung 5.17: Falsch negative Testergebnisse von Fehlertest mit Stichprobentest

5.1.3 Verlässliche DHT-Anwendung

Nachdem das Chord Protokoll und seine Erweiterungen in ihrer Funktionsweise durch Messungen belegt sind, erfolgt nun die Evaluation der darauf aufbauenden verlässlichen DHT-Anwendung. Hierbei wird wiederum zwischen Messungen in einem statischen und einem dynamischen Netzwerk unterschieden, wobei die Messungen im statischen Netzwerk wiederum einer idealisierten Umgebung entsprechen.

5.1.3.1 Statisches Netzwerk

Für die statischen Messungen wird ein verlässliches Chord Netzwerk mit der für die Messung angegebenen Knotenanzahl und dem entsprechenden Anteil bösartiger Knoten aufgebaut. Im Gegensatz zu vorhergehenden Messungen fungierte nun jeder Knoten zusätzlich als DHT-Knoten der für PACS entwickelten DHT-Anwendung, die in Kapitel 3.5.7.8 vorgestellt wird.

Nach dem Aufbau des Netzwerks werden zunächst 10000 Datenobjekte in der DHT gespeichert. Hierzu tätigen zufällig ausgewählte gutartige Knoten $put(key, priv)$ Aufrufe. $priv$ steht hierbei für das zufällig erzeugte Datenobjekt und key für den ebenfalls zufällig erzeugten Schlüssel, unter dem das Datenobjekt abgelegt werden soll. Aus dem key wird durch die Funktion $ChordID(h(key)) = id\_key$ die ChordID des Datenobjektes berechnet, die den Speicherort des Datenobjekts im Netzwerk festlegt.

Nach der Speicherung der Objekte erfolgt die Abfrage derselben. Hierzu führen wiederum zufällig ausgewählte gutartige Knoten 10000 Datenabfragen aus. Die abzufragenden Datenobjekte werden hierbei zufällig aus der Menge der gespeicherten Datenobjekte ausgewählt. Die vorliegenden Messungen werden zudem zehnmal mit verschiedenen Zufallszahlen wiederholt und deren Durchschnitt berechnet.

In dem der DHT zugrundeliegenden Chord Netzwerk verhalten sich bösartige Peers wie zuvor im Kapitel 5.1.2 erläutert. Auch in der DHT-Anwendung ist es ihr Ziel, andere Knoten zu manipulieren. Bei *get* Anfragen werden Datenobjekte, die auf bösartig kooperierenden Peers gespeichert sind, deshalb von ihnen entweder nie, nicht immer oder beliebig verfälscht

zurückgeliefert. Ihr Verhalten richtet sich hierbei danach, was der Gruppe der bösartig kooperierenden Peers in der jeweiligen Situation am meisten nützt. Bei der gezeigten Anfrage nach einem Datenobjekt ist dies die Rückgabe eines verfälschten Ergebnisses. Eine Nichtbeantwortung der Antwort führt nämlich unweigerlich zu Anforderung einer SKM und so würde die Manipulation verhindern. Bei *put* Anfragen werden die bösartigen Peers das Datenobjekt wie gefordert speichern. Sie werden es aber nur auf andere bösartig kooperierende Peers ihrer Replikationsgruppe übertragen. Das Datenobjekt bleibt hierdurch innerhalb der Gruppe der bösartigen Peers und wird von ihnen kontrolliert, sofern die Manipulation nicht erkannt wird. Bei der Stichprobenprüfung der korrekten Speicherung ist so ein gutartiger Peer in der Stichprobe erforderlich, um die Manipulation aufzudecken.

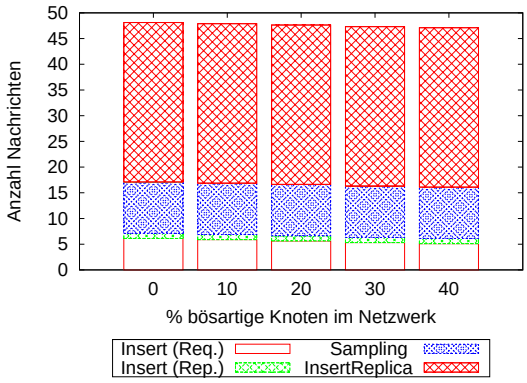
In Abbildung 5.18 ist exemplarisch die Anzahl der Nachrichten für die verschiedenen Nachrichtenarten für die *put* Anfragen in einem Netzwerk mit 10000 Knoten und einem Anteil von 0 bis 50% bösartiger Knoten aufgeschlüsselt. Die Messungen für die Netzwerkgrößen 1000 und 5000 befinden sich im Anhang E.

Wie in den Messungen in Abbildung 5.18(a) zu sehen, ist der Aufwand für das normale Routing unabhängig von der Anzahl bösartiger Knoten. Normales Routing bedeutet hier, dass für das Speichern eines Datenobjektes nur das verbesserte Chord Routing von PACS zum Einsatz kommt. Entsprechend ist der Fehlertest inklusive den Stichproben mit der zurückgelieferten AKM dieses Chord basierten Routings negativ. Eine Anforderung einer SKM durch den einfügenden Peer ist deshalb nicht nötig. Für die Messung mit 50% bösartigen Knoten fehlt die Darstellung, da bei den Messungen keine ausreichende Zahl erfolgreicher Anfragen mit normalem Routing ermittelt werden konnte. Da PACS 32 Replikate speichert, stellen die InsertReplica-Nachrichten hierbei den Hauptanteil der Nachrichten dar.

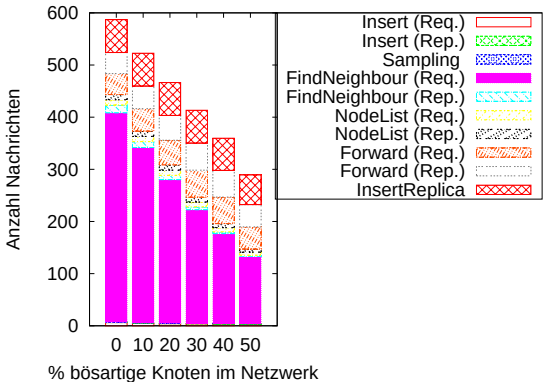
In Abbildung 5.18(b) ist die Anzahl Nachrichten für den Fall eines positiven Fehler- oder Stichprobentests dargestellt. Durch sicheres Routing wird eine SKM angefordert und die Speicherung des Datenobjekts in den Knoten dieser Menge durch den einfügenden Peer selbst veranlasst. Wie ersichtlich, ist die dominierende Größe der Nachrichten hierbei die Anzahl FindNeighbour-Request-Nachrichten. Dies sind exakt die Nachrichten, die letztlich über den Erfolg des redundanten Routing Algorithmus entscheiden. Die Anzahl der FindNeighbour-Nachrichten nimmt mit zunehmendem Anteil bösartiger Peers ab, da diese FindNeighbour-Nachrichten nicht weiterleiten.

In Abbildung 5.18(c) ist der Gesamtdurchschnitt aller *put* Anfragen dargestellt. Es handelt sich hierbei um die anteiligen Mischungen zwischen normalem und sicherem Routing, die in den Abbildungen 5.18(a) und 5.18(b) dargestellt sind. Die Gesamtanzahl der Nachrichten nimmt mit ansteigendem Anteil bösartiger Peers zu, um ab 30% bösartiger Peers wieder abzufallen. Grund hierfür ist, dass schon bei einem Anteil von 30% bösartiger Peers kaum *put* Anfragen mit normalen Routing erfolgreich sind. Entsprechend ist die Erhöhung des Anteils an sicheren Routing Anfragen gering. Hingegen reduziert sich die Anzahl der FindNeighbour-Nachrichten beim sicheren Routing durch den höheren Anteil bösartiger Nachrichten (wie aus Abbildung 5.18(b) ersichtlich), was insgesamt zur einer Abnahme der Nachrichtenmenge führt.

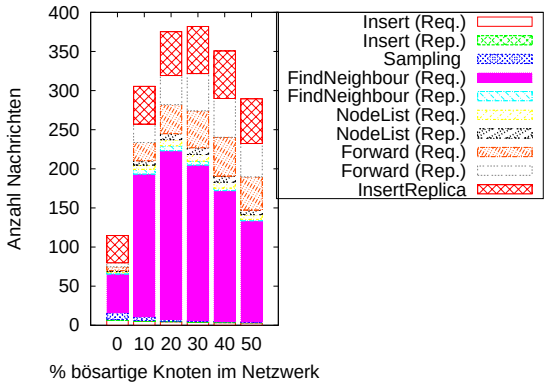
Der Zusatzaufwand, der durch das sichere Routing entsteht, wird von der Fehlerkontrolle effektiv eingedämmt. So ist der Gesamtdurchschnitt nur ca. 25% höher, als der Aufwand, der durch ausschließliches normales Routing entsteht. D.h. lediglich beim Vorhandensein von bösartigen Knoten kommt es zu der dargestellten wesentlichen Erhöhung der Nachrichten, die dann aber zur Sicherstellung eines funktionierenden Netzwerks elementar ist. In Abbildung 5.19(a) ist die Zuverlässigkeit der *put* und *get* Anfragen dargestellt. Diese ist bis zu einem



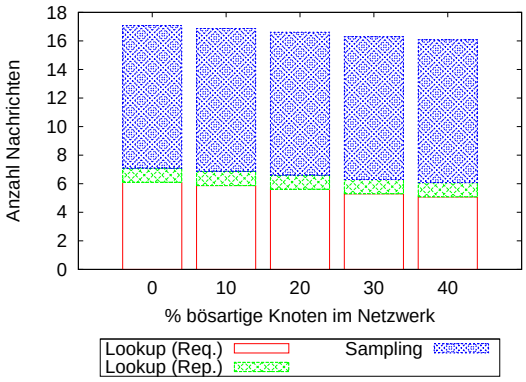
(a) Normales Routing *put*



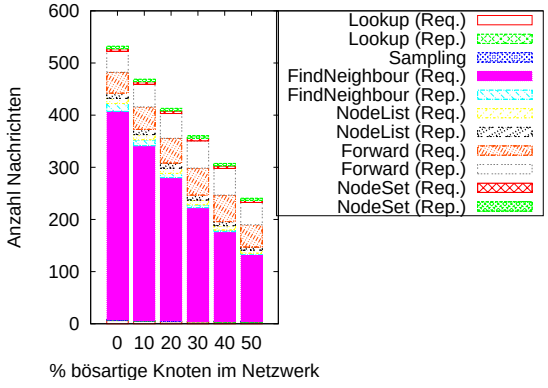
(b) Sicheres Routing *put*



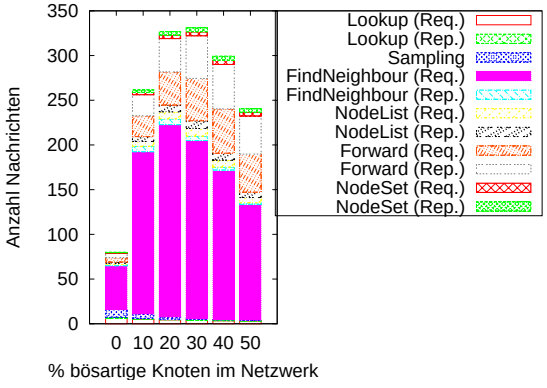
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*

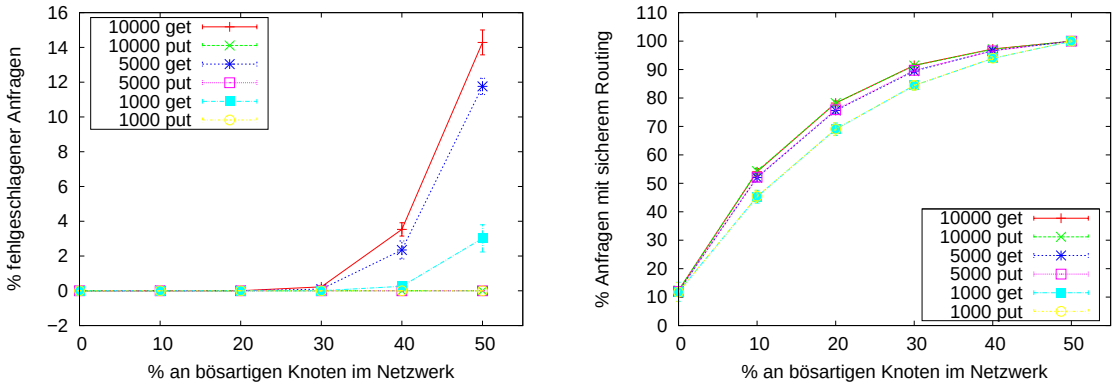


(e) Sicheres Routing *get*



(f) Gesamtdurchschnitt *get*

Abbildung 5.18: Nachrichtenanzahl bei *put* und *get* Anfragen im statischen Netzwerk



(a) Fehlerrate

(b) Anteil sicheres Routing

Abbildung 5.19: Fehlerrate und Anteil sicheres Routing bei *put* und *get* Anfragen

Anteil von 30% bösartiger Knoten gegeben, denn der Anteil an fehlgeschlagenen Anfragen ist unterhalb von 0,25%. Danach steigt die Fehlerrate bei *get* Anfragen auf bis zu 4% an. In Abbildung 5.19(b) wird der Anteil der Anfragen mit positivem Fehlertest oder Stichprobentest dargestellt, bei denen folglich sicheres Routing verwendet wird. Die Kurve nähert sich, wie aus den vorherigen Messungen schon vermutet, relativ schnell der 100% Marke an. Insbesondere tritt ab 30% keine signifikante Erhöhung mehr ein.

In Abbildung 5.18 wird die Anzahl der Nachrichten aufgeschlüsselt auf ihre spezifische Ausprägung für ein 10000 Knoten Netzwerk bei *get* Anfragen dargestellt. Die entsprechenden Messungen für die Netzwerkgrößen 1000 und 5000 befinden sich im Anhang E.

Wie bei den *put* Anfragen, werden auch die *get* Anfragen aufgeteilt in Anfragen, die nur das erweiterte Chord basierte Routing verwendeten (Abbildung 5.18(d)) und solche, die sicheres Routing benutzen, um die Anfrage erfolgreich zu beantworten (Abbildung 5.18(e)). In Abbildung 5.18(f) schließlich wird der Gesamtdurchschnitt dargestellt, der sich aus der anteiligen Mischung aus normalem und sicherem Routing ergibt. Das normale Routing verändert sich mit zunehmendem Anteil bösartiger Peers kaum. Auf eine Darstellung für 50% bösartige Peers wird verzichtet, da bei den Messungen zu wenige Anfragen vorhanden waren. Selbst bei einem 10000 Knoten Netzwerk und der sich dadurch ergebenden Anzahl an Routingschritten ist der Hauptanteil der Nachrichten dem Sampling zuzuordnen. Bei den *get* Anfragen mit sicherem Routing ist derselbe Effekt wie bei entsprechenden *put* Anfragen zu beobachten. Beim Hauptteil der Nachrichten handelt es sich um FindNeighbour-Nachrichten, deren Anzahl mit zunehmendem Anteil bösartiger Knoten abnimmt. Da bei *get* Anfragen die InsertReplica-Nachrichten wegfallen, ist die Gesamtzahl der Nachrichten beim sicheren Routing geringer als bei *put* Anfragen.

Beim Gesamtdurchschnitt der *get* Anfragen bestätigt sich erneut die Effizienz des Fehlertests. So wird der Aufwand für das sichere Routing nur dann nötig, sobald sich bösartig kooperierende Peers im Netzwerk befinden. Der Aufwand, den das sichere Routing bei 0% bösartigen Peers im Netzwerk erzeugt, ist bei *get* Anfragen erheblich höher. Anstatt den bei ausschließlich normalem Routing gemessenen ca. 17 Nachrichten sind aufgrund des sicheren Routings im Durschnitt ca. 80 Nachrichten pro *get* Anfrage zu verzeichnen. Dieser erhöhte Aufwand wird durch die gewonnene Sicherheit gerechtfertigt, auch bei bösartigen Knoten im Netzwerk die Daten auffinden zu können.

5.1.3.2 Dynamisches Netzwerk

Für die vorliegenden Messungen wird alle fünf Simulationsschritte eine Veränderung im Netzwerk vorgenommen, wobei parallel dazu zunächst 10000 *put* Anfragen und anschließend 10000 *get* Anfragen bearbeitet werden. Das Netzwerk besitzt somit eine Änderungsfrequenz von fünf Simulationsschritten. Die ausführenden Knoten werden zufällig aus den gutartigen Knoten ausgewählt, wobei darauf geachtet wird, dass sie nicht während der Ausführung der Anfrage das Netzwerk verlassen. Das Vorgehen erfolgt ansonsten analog zu jenem der statischen Messung.

In Abbildung 5.20 ist exemplarisch die Anzahl der Nachrichten für die verschiedenen Nachrichtenarten für *put* Anfragen und *get* Anfragen im dynamischen Netzwerk mit 10000 Knoten, mit einer Änderungsfrequenz von fünf Simulationsschritten und einem Anteil von 0 bis 50% bösartiger Knoten im Netzwerk dargestellt. Die Messungen für die Netzwerkgrößen 1000 und 5000 befinden sich im Anhang E.

Die Ergebnisse entsprechen im Wesentlichen jenen der statistischen Messungen und bedürfen deshalb keiner erneuten Erläuterung. Die Dynamik ändert die Anzahl der Nachrichten für eine Anfrage nur marginal um höchstens eine Nachricht. Gleiches gilt für den Anteil sicherer Routing Anfragen. Wie in Abbildung 5.21(b) ersichtlich, erhöht sich deren Anteil lediglich bei 5000 und 10000 Knoten um ein bis zwei Prozent gegenüber den statischen Messungen.

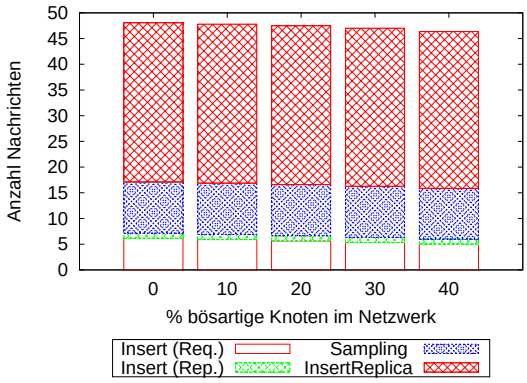
Anders stellt sich dies für die Zuverlässigkeit dar (siehe Abbildung 5.21(a)). Wie erwartet, verschlechterte sich die Zuverlässigkeit durch die Dynamik, je kleiner das Netzwerk ist. Dennoch ist auch dort bei 30% bösartigen Peers die Zuverlässigkeit für *put* und *get* Anfragen größer als 99%.

Der Aufwand, den ein beitretender Knoten im Netzwerk erzeugt, ist in Abbildung 5.22 für verschieden große Netzwerke dargestellt. Wichtig ist hierbei die Menge an Nachrichten, die erzeugt wird, um die SKM für einen beitretenden Knoten zu ermitteln. Jeder Knoten wird fünf Knoten nach einer SKM fragen, um sicherzugehen, dass er nicht zufällig einen bösartigen Knoten anfragt. Entsprechend beträgt die Anzahl Nachrichten für einen Beitritt eines Knotens zum Netzwerk das Fünffache der hier dargestellte Menge.

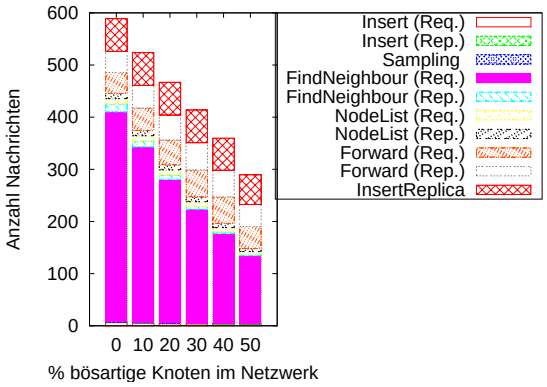
5.1.3.3 Zusammenfassung der Ergebnisse für die verlässliche DHT

Aus den Simulationen und Messungen geht eindeutig hervor, dass die für PACS entwickelte verlässliche DHT bis zu einem Anteil von 30% bösartig kooperierender Peers die ihr anvertrauten Daten zuverlässig verwaltet. Der maximale Aufwand zur Erreichung dieser Zuverlässigkeit ist für *put* Anfragen mit bis zu 600 Nachrichten und 530 Nachrichten für *get* Anfragen (jeweils in einem 10000 Knoten Netzwerk beträchtlich. Dieser maximale Aufwand fällt jedoch aufgrund des Fehlertests nur in wenigen Fällen an. So liegt bei einem Netzwerk ohne bösartige Knoten der Aufwand der verlässlichen Hashtabelle mit ca. 110 Nachrichten für *put* und ca. 75 Nachrichten für *get* Anfragen im Rahmen, wenngleich dies im Vergleich zu den Werten des normalen Routings einer Verdopplung bei *put* bzw. Verdreifachung bei *get* entspricht.

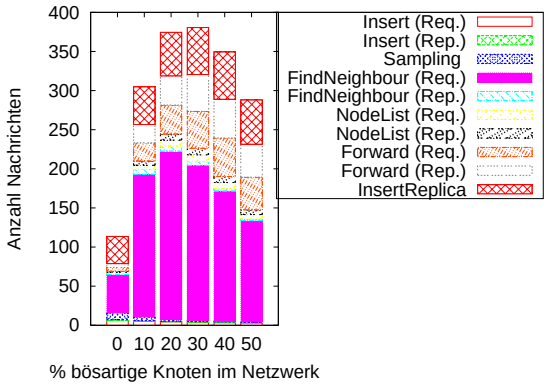
Höhere Kosten verursacht die verlässliche DHT falls tatsächlich bösartig kooperierende Peers im Netzwerk vorhanden sind. In Anbetracht der Alternative einer nicht funktionierenden DHT ist dieser erhöhte Aufwand aber ebenfalls akzeptabel. Ein Hemmschuh für die verlässliche DHT ist die hohe Einstiegshürde durch die Anforderung von fünf $SKMs$ beim Beitritt zum Netzwerk, was einem Aufwand von bis zu 2500 Nachrichten entspricht. Ein häufiges Verlassen und Beitreten von Knoten zum Netzwerk belastet deshalb das Netzwerk enorm.



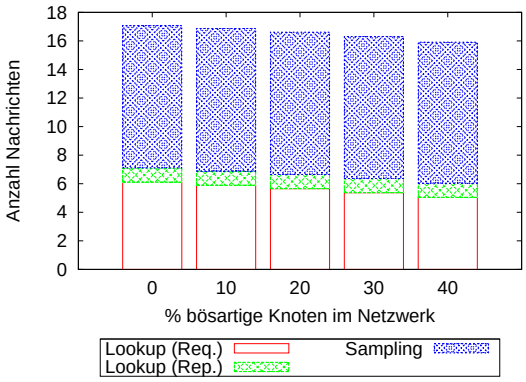
(a) Normales Routing *put*



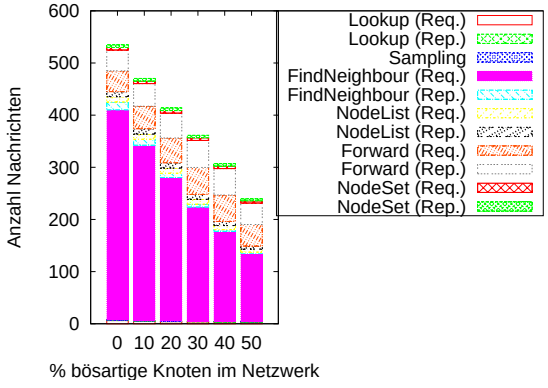
(b) Sicheres Routing *put*



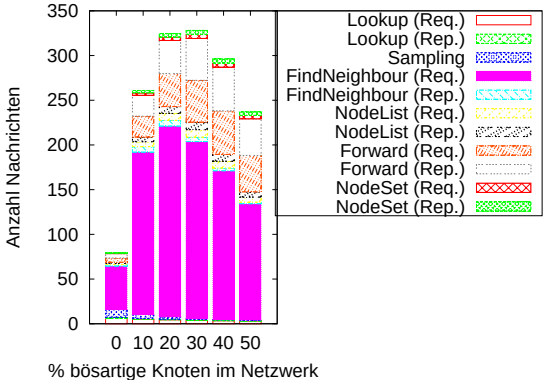
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*

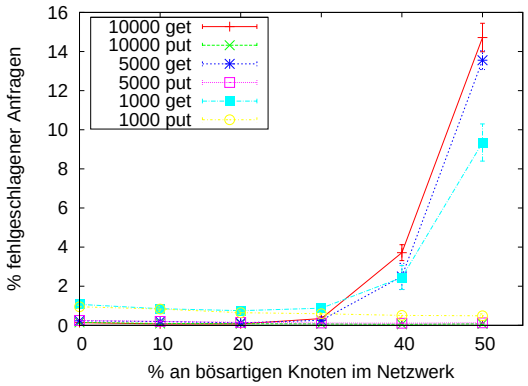


(e) Sicheres Routing *get*

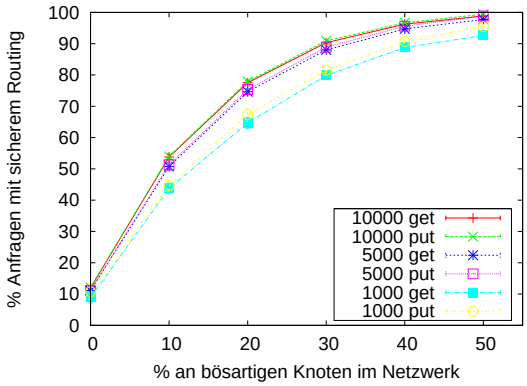


(f) Gesamtdurchschnitt *get*

Abbildung 5.20: Nachrichtenanzahl bei *put* und *get* Anfragen im dynamischen Netzwerk

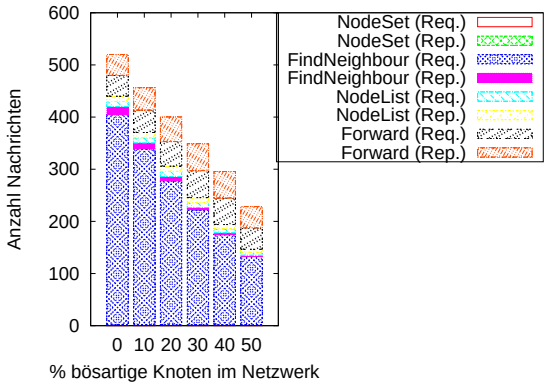


(a) Fehlerrate

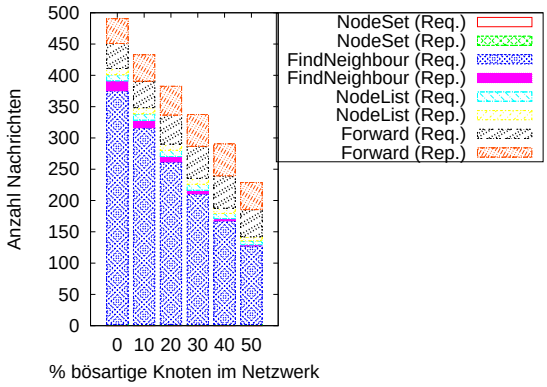


(b) Anteil sicheres Routing

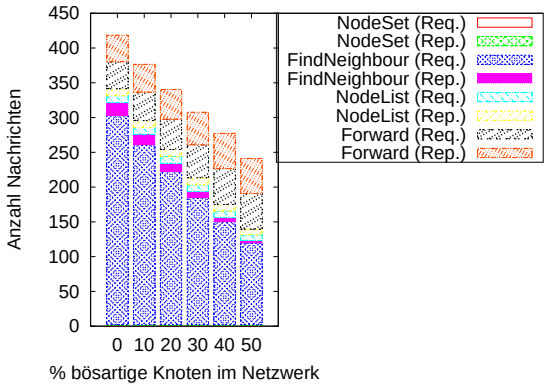
Abbildung 5.21: Fehlerrate und Anteil sicheres Routing bei *put* und *get* Anfragen



(a) Netzwerk mit 10000 Knoten



(b) Netzwerk mit 5000 Knoten



(c) Netzwerk mit 1000 Knoten

Abbildung 5.22: Aufwand Erzeugung SKM für den Beitritt eines Knotens

Aber auch der beitretende Peer muss beträchtliche Vorleistungen zum Beitritt erbringen, wie das Anfordern von Zertifikaten und Anfragen von Bootstrap-Knoten. Dieser Aufwand ist jedoch ohne Alternative, um auch bei 30% bösartig kooperierenden Peers eine Manipulation des beitretenden Peers zu verhindern. Er unterbindet aber letztendlich den Einsatz der verlässlichen DHT in sehr dynamischen Umgebungen.

5.1.4 Zusammenfassung der Ergebnisse

Die Funktionstüchtigkeit von strukturierten P2P-Netzen in Umgebungen, selbst mit geringen Anteilen bösartig kooperierender Peers, ist nicht gewährleistet, wie eindeutig durch Messungen belegt wurde.

Die in PACS vorgenommen Erweiterungen und Anpassungen am Chord Protokoll und die Funktionstüchtigkeit der darauf aufbauenden verlässlichen DHT wurden durch Messungen dargelegt. Diese Erweiterungen und die verlässliche DHT-Anwendung verursachen hierbei gegenüber einer normalen DHT einen enormen Mehraufwand und sind in ihrem Anwendungsbereich dadurch eingeschränkt.

Als Alternative bietet sich der Einsatz des unstrukturierten P2P-Netzwerks mit aktiver Replikation an, das nun im Folgenden erläutert wird.

5.2 Ergebnisse des unstrukturierten P2P-Netzwerks

In diesem Kapitel werden die Ergebnisse des unstrukturierten P2P-Netzwerkspeichers mit definierter Replikationsgruppe gezeigt, wie er in Kapitel 3.5.6 erläutert wird. Aufgrund des einfacheren Aufbaus des P2P-Netzwerks sind die Simulationen und Messungen weniger umfangreich, als bei der verlässlichen DHT. Zunächst wird die generelle Zuverlässigkeit des Flooding Algorithmus dargestellt. Anschließend wird der Einfluss der Anzahl der Replikationsgruppen untersucht. Daran schließen sich die Messungen über die Auswirkungen von bösartigen Knoten im Netzwerk an. Abschließend erfolgt die Darstellung der Zuverlässigkeit von *put* und *get* Anfragen in einem solchen Netzwerk.

5.2.1 Zuverlässigkeit des Flooding Algorithmus

Beim im unstrukturierten P2P-Netzwerk von PACS eingesetzten Flooding Algorithmus ist die maximale Anzahl an Weiterleitungen die entscheidende Größe. Diese wird durch das Attribut $maxHopCount$ in den Flooding-Nachrichten bestimmt. Die Erreichbarkeit der Knoten im Netzwerk wird deshalb mit verschiedenen $maxHopCount$ und Netzwerkgrößen gemessen.

Für die Simulationen wird zunächst wiederum ein Netzwerk mit der angestrebten Größe und dem vorgegebenen Anteil bösartiger Knoten erzeugt. Da in einem unstrukturierten Netzwerk keine Bedingungen an die Fingertabelleneinträge der Knoten gestellt werden, werden diese Einträge zufällig aus den vorhandenen Knoten im Netzwerk ausgewählt. Anschließend wird die Erreichbarkeit der Knoten ermittelt. Hierzu werden aus der Menge zufällig Knoten ausgewählt, die versuchen, andere zufällig ausgewählte Knoten zu erreichen. In einer Simulation wird dieser Vorgang 10000-mal wiederholt. Die dargestellten Ergebnisse entsprechen dem Durchschnitt aus zehn Simulationsdurchläufen mit unterschiedlicher „Seed“ des Zufallszahlengenerators.

Für die in Abbildung 5.23 dargestellten Ergebnisse eines statischen Netzwerks entspricht $maxHopCount = 2$; der Anteil bösartiger Knoten wird variiert. Das Verhalten der bösartigen

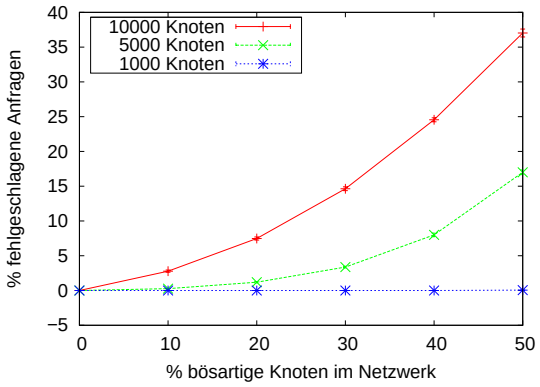


Abbildung 5.23: Erreichbarkeit der Knoten mit $maxHopCount = 2$

Peers besteht beim Flooding Algorithmus darin, Nachrichten nicht weiterzuleiten und so die Erreichbarkeit der Knoten zu verhindern. Auf eine Darstellung der Erreichbarkeit der Knoten in einem dynamischen Netzwerk kann an dieser Stelle verzichtet werden, da in einem unstrukturierten Netzwerk durch das Hinzukommen oder Weggehen eines Peers nicht das gesamte Netzwerk betroffen ist, sondern nur jene Knoten, die einen Fingereintrag zu diesem Knoten besitzen. Entsprechend sind die Auswirkungen der Dynamik auf die Funktionsweise des Netzwerks wesentlich geringer.

Abbildung 5.23 belegt, dass alle Knoten im Netzwerk innerhalb von zwei Routingschritten erreicht werden können, falls sich nur gutartige Knoten im Netzwerk befinden. Mit zunehmendem Anteil bösartiger Knoten erhöht sich die Fehlerrate. Insbesondere steigt diese, je größer das Netzwerk ist. Während bei einer Netzwerkgröße von 1000 die Fehlerrate bei 0% verbleibt, steigt sie bei nur 10% bösartiger Knoten bei 5000 Knoten auf ca. 0,25% und bei 10000 Knoten schon auf ca. 2,8% an.

Wie bei der Messung mit 1000 Knoten ersichtlich, ist bei einer im Vergleich zur Netzwerkgröße ausreichenden Anzahl redundanter Nachrichten, selbst bei einem Netzwerk mit 50% bösartigen Peers, eine verlässliche Erreichbarkeit aller Knoten gegeben. Der Flooding Algorithmus arbeitet demnach bei entsprechender Einstellung trotz bösartiger Knoten im Netzwerk zuverlässig. Dessen Aufwand steigt mit zunehmendem $maxHopCount$ und $FINGERSIZE$ im schlechtesten Fall exponentiell, denn die maximale Anzahl der Nachrichten ergibt sich durch $FINGERSIZE^{maxHopCount}$. Die hier gezeigten Messungen machen noch keine Aussage über die Auffindbarkeit von gespeicherten Daten, da diese repliziert auf den verschiedenen Knoten vorliegen. Der Einfluss dieser Replikation wird deshalb im folgenden Abschnitt genauer untersucht.

5.2.2 Einfluss der Anzahl der Replikationsgruppen

Für die sichere Speicherung und Abfrage von Datenobjekten besitzt jeder Knoten eine Replikationsgruppe und ist eventuell selbst Mitglied in einer oder mehreren Replikationsgruppen. Jeder Knoten kann seine eigene Replikationsgruppe besitzen, allerdings ist dies nicht zwingend erforderlich. Es ist üblich, dass ein Knoten, der selbst Mitglied in einer Replikationsgruppe ist, diese Gruppe auch als seine eigene Replikationsgruppe wählt. Eine definierte Gruppe von Knoten kann dadurch als Replikationsgruppe für mehrere Knoten dienen.

In einem ersten Schritt wird deshalb die Auswirkung der Anzahl an Replikationsgruppen

auf die Zuverlässigkeit von Datenspeicherungen und Datenanfragen untersucht. Zudem wird der Aufwand gemessen, den Anfragen oder Änderungen im Netzwerk verursachen.

Im Gegensatz zu den vorhergegangenen Messungen wird hierzu die Erreichbarkeit von gespeicherten Datenobjekten unter verschiedenen Replikationsgruppen ermittelt. Die Erzeugung der Replikationsgruppen erfolgt nachdem das Netzwerk seine endgültige Größe erreicht hat. Die Replikationsgruppen werden hierbei zufällig aus der Menge aller Knoten ausgewählt, um so eine gleichmäßige Verteilung der Knoten in den Gruppen zu gewährleistet. Anschließend wird jedem Knoten eine zufällig ausgewählte Replikationsgruppe zugewiesen. Kommen in einem dynamischen Netzwerk neue Knoten hinzu, erhalten diese, nachdem sie dem Netzwerk vollständig beigetreten sind, ebenfalls eine zufällig ausgewählte Replikationsgruppe zugewiesen.

Ausfallende Knoten der Replikationsgruppe werden gemäß dem in Kapitel 3.5.6.4 beschriebenen Protokoll ersetzt. Die verbleibenden Gruppenmitglieder tauschen hierbei den ausgefallenen Knoten mit einem zufällig ausgewählten Knoten, sodass auch hier eine gleichmäßige Verteilung gewährleistet bleibt.

Die Messungen zum Einfluss der Replikationsgruppe beschränken sich auf die Darstellungen des Szenarios mit 30% bösartig kooperierenden Knoten im Netzwerk. Es handelt sich bei den dargestellten Ergebnissen um den Durchschnitt von mindestens zwei Simulationsdurchläufen beim statisches Netzwerk und zehn Simulationsdurchläufen beim dynamischen Netzwerk mit unterschiedlicher Initialisierung des Zufallszahlengenerators.

Bösartig kooperierende Knoten verhalten sich bei *get* Anfragen analog zur DHT-Anwendung und antworten entsprechend bei den hier getätigten Messungen nur mit einem manipulierten Datenobjekt. Bei *put* Anfragen speichern bösartig kooperierende Knoten das Datenobjekt zwar, leiten es aber bei Gruppenänderungen nicht an neue Gruppenmitglieder weiter. Mit diesem Verhalten erschweren sie den gutartigen Peers die zuverlässige Verwaltung der Datenobjekte im Netzwerk am stärksten.

Für den Simulationsaufbau werden zunächst 10000 Datenobjekte bei zufällig ausgewählten Peers gespeichert. Hierbei werden diese durch den speichernden Peer in dessen Replikationsgruppe repliziert. Anschließend erfolgen 10000 Anfragen für die nun gespeicherten Daten von zufällig ausgewählten Peers. Die Ergebnisse für Netzwerke mit 30% bösartigen Knoten und $maxHopCount$ von zwei sind in Abbildung 5.24 zu sehen. Sie zeigen, dass mit $maxHopCount$ von zwei die Zuverlässigkeit trotz Replikation nur bis zu einer Netzwerkgröße von 5000 gegeben ist. Eine Erhöhung von $maxHopCount$ führt jedoch zu einer unverhältnismäßigen Steigerung um Faktor 32 der Nachrichtenanzahl und wird deshalb nicht weiter verfolgt. Abbildung 5.24 zeigt neben den statischen Messungen auch die Simulationsergebnisse mit einem dynamischen Netzwerk. Die Änderungsfrequenz beträgt hierbei fünf Simulationsschritte. Während bei 1000 Knoten die Ergebnisse beim statischen und dynamischen Netzwerk praktisch identisch sind (Abweichung maximal 0,05%), beträgt die Abweichung bei 5000 und 10000 Knoten maximal 1% bzw. 1,5%. Dies bestätigt die zuvor gemachte Aussage, dass die Auswirkungen der Dynamik im unstrukturierten P2P-Netzwerk sehr gering sind. Interessanterweise sind die Ergebnisse sogar leicht besser als im statischen Fall. Dies lässt sich durch die ungleiche Verteilung der Knoten in den Replikationsgruppen und Fingertabellen erklären, die durch die Dynamik entsteht. Knoten, die länger im Netzwerk sind, haben häufiger die Chance als Ersatz für weggehende Knoten in Replikationsgruppen und Fingertabellen anderer Peers berufen zu werden. Hierdurch ergibt sich eine zumindest zeitweise ungleiche Verteilung. Da diese zwischen Fingertabellen und Replikationsgruppe korreliert, kommt es zu den hier gezeigten besseren Ergebnissen. Dass dieser Effekt nur temporär auftritt, ist an den Ergebnissen mit

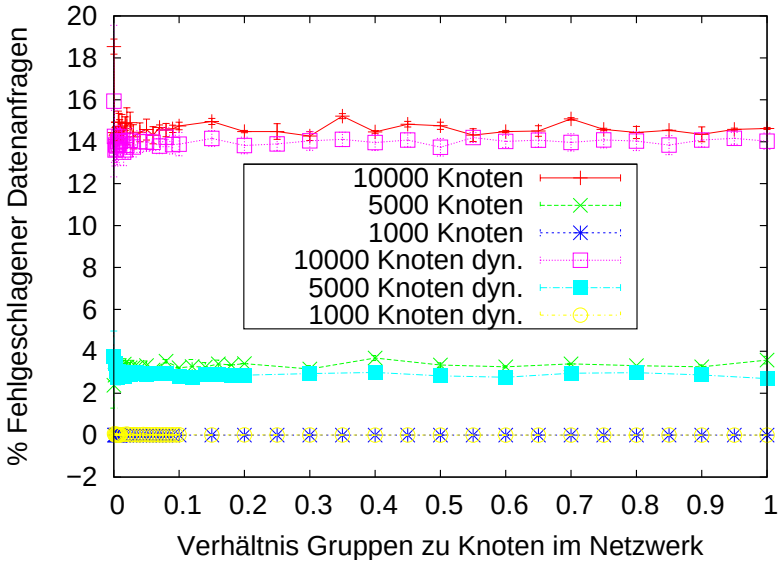


Abbildung 5.24: *get* Anfragen bei unterschiedlicher Anzahl Replikationsgruppen (30% bösartige Knoten im Netzwerk)

1000 Knoten zu sehen. Aufgrund der während der Messung vorgenommenen 4000 Änderungen im Netzwerk verlässt jeder Knoten das Netzwerk durchschnittlich zweimal. Die ungleiche Verteilung tritt deshalb hier nicht auf und so sind die Ergebnisse mit den statischen Ergebnissen nahezu identisch. Bei 5000 und 10000 Knoten verlassen hingegen während der Messung nur 40% bzw. 20% Knoten das Netzwerk weshalb dort der Effekt zu beobachten ist.

Entscheidend bei den hier dargestellten Messungen ist, dass die Anzahl der Replikationsgruppen keinen Einfluss auf die Zuverlässigkeit des Netzwerkes hat. Für die jeweilige Netzwerkgröße pendelt sich die Fehlerrate auf einem charakteristischen Niveau ein und ist somit unabhängig von der Anzahl der Replikationsgruppen.

Die Gruppenanzahl im Netzwerk hat jedoch Einfluss auf Teilbereiche des P2P-Protokolls. So ist es erforderlich, dass sich die Gruppenmitglieder gegenseitig dahingehend kontrollieren, ob alle Gruppenmitglieder noch vorhanden sind. Hierzu senden sie Alive-Nachrichten an die anderen Gruppenmitglieder. Je mehr Gruppen im Netzwerk vorhanden sind, desto mehr solcher Nachrichten müssen versendet werden. Dies ist in Abbildung 5.25 für die verschiedenen Netzwerkgrößen mit unterschiedlicher Anzahl Gruppen dargestellt. Wie ersichtlich, nimmt die Anzahl der Nachrichten mit der Anzahl der Gruppen im Netzwerk zu. Je mehr Gruppen sich im Netzwerk befinden, desto mehr Alive-Nachrichten werden erzeugt. Auch die Anzahl der fehlgeschlagenen Alive-Anfragen steigt kontinuierlich. Bis zu einem Verhältnis Gruppe zu Netzwerkgröße von 0,03 ist dies durch die gestiegene Wahrscheinlichkeit zu erklären, dass ein Knoten in einer Replikationsgruppe ist. Danach sind mehr Knoten in mehreren Gruppen, weshalb durch das Ausscheiden eines Knotens gleich mehrere Gruppen verändert werden.

Die Anzahl der Replikationsgruppen bestimmt somit die Verwaltungskosten des unstrukturierten P2P-Netzwerks mit aktiver Replikation. Diese steigen mit der Anzahl der Gruppen linear an.

Bei einer Gruppenänderung läuft das in Kapitel 3.5.6.4 beschriebene Protokoll ab. In Abbildung 5.26(a) ist der Aufwand dargestellt, den der Ablauf dieses Protokolls bei unter-

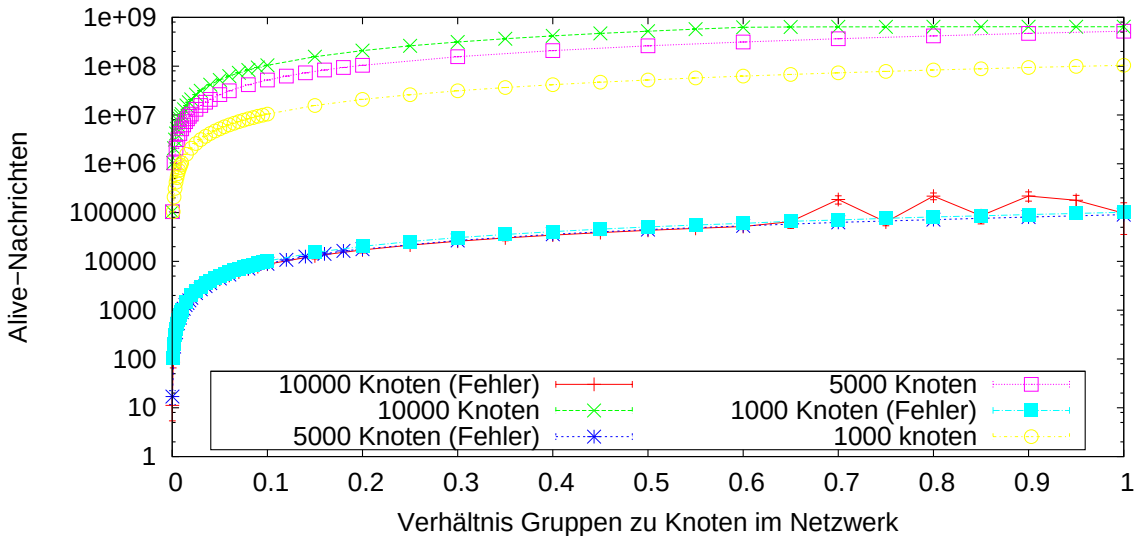


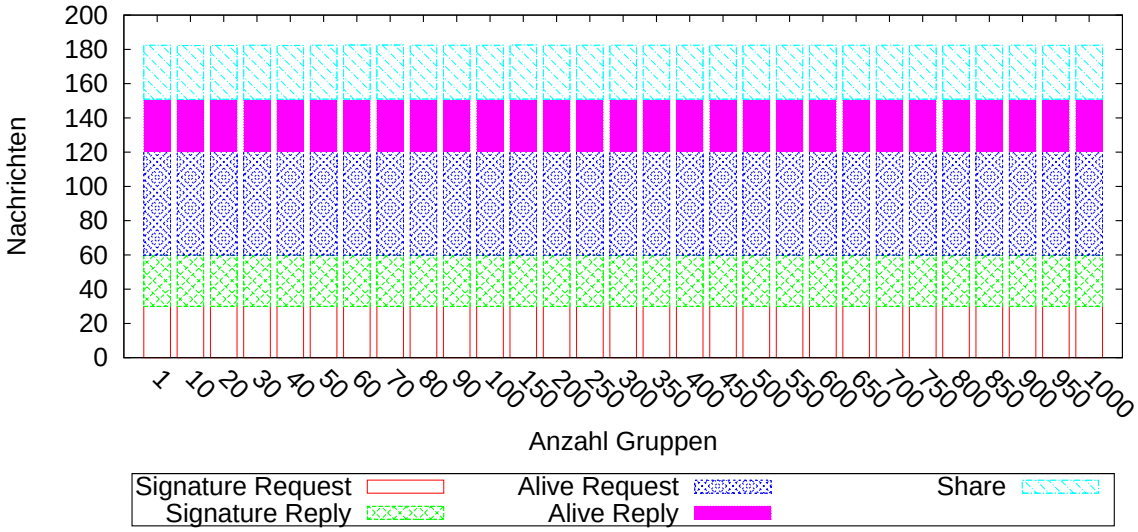
Abbildung 5.25: Alive-Nachrichten bei unterschiedlicher Anzahl Replikationsgruppen (30% bösartige Knoten im dynamischen Netzwerk)

schiedlicher Anzahl von Replikationsgruppen im Netzwerk erzeugt. Wie ersichtlich ändert sich dieser Aufwand für eine einzelne Gruppenänderung nicht. Bei den hier dargestellten Ergebnissen handelt es sich um ein Netzwerk mit 1000 Knoten und 30% bösartigen Knoten. Da das Protokoll unabhängig von der Netzwerkgröße ist, wird auf eine Darstellung der Messergebnisse für 5000 und 10000 Knoten Netzwerke verzichtet. Die übrigen Gruppenmitglieder werden vom Manager Peer über die Abwesenheit eines Gruppenmitglieds durch Signature-Nachrichten unterrichtet. Diese senden dann zur Überprüfung der Angaben Alive-Anfragen an den abwesenden Peer und das neu aufzunehmende Gruppenmitglied. Da lediglich das neu aufzunehmende Gruppenmitglied auf diese Anfragen antwortet, ist die Anzahl Alive-Antworten nur halb so groß. Anschließend erfolgt die Übermittlung der Share-Nachrichten. Nicht dargestellt ist hier die Übermittlung der Replikate durch den Manager Peer an das neue Gruppenmitglied, denn diese ist von der Anzahl der gespeicherten Datenobjekte abhängig. Wie in Abbildung 5.26(b) zu sehen, ist der Aufwand für eine Gruppenänderung vom Anteil bösartiger Peers im Netzwerk unabhängig.

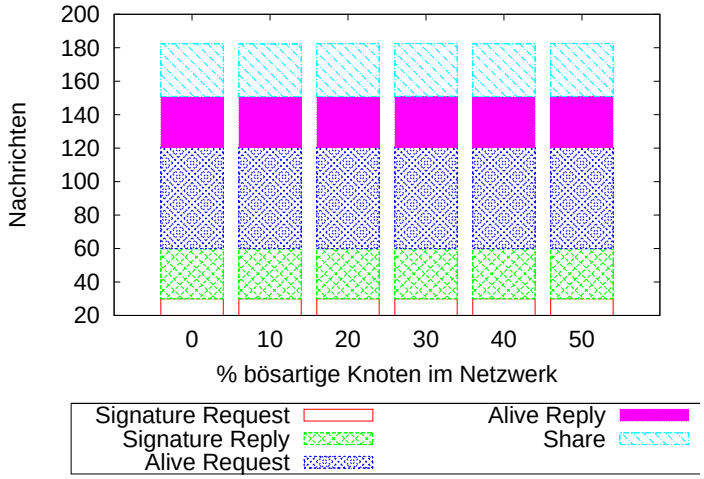
Da die Anzahl der Gruppen im Netzwerk lediglich Auswirkungen auf den Verwaltungsaufwand im Netzwerk hat, und nicht auf dessen Funktionsfähigkeit und Zuverlässigkeit, wird für die nachfolgenden Messungen die Anzahl der Gruppen auf 30, 150 bzw. 300 für die entsprechende Netzwerkgrößen von 1000, 5000 und 10000 Knoten festgelegt. Bei dieser Anzahl an Gruppen für die Netzwerkgrößen ist jeder Knoten in etwa einer Replikationsgruppe enthalten. Mit diesen Vorgaben wird nun das Verhalten des unstrukturierten P2P-Netzwerks mit definierten Replikationsgruppen von PACS genauer untersucht.

5.2.3 Aktive Replikation mit bösartigen Knoten

Der entscheidende Faktor für die Zuverlässigkeit des unstrukturierten P2P-Netzwerks mit aktiver Replikation von PACS ist der Anteil bösartig kooperierender Knoten im Netzwerk. Mit der zuvor festgelegten Anzahl an Replikationsgruppen werden deshalb Messungen mit



(a) Unterschiedliche Gruppen bei 30% bösartigen Peers



(b) 300 Gruppen mit unterschiedlicher Anzahl bösartiger Knoten

Abbildung 5.26: Aufwand einer Gruppenänderung bei 1000 Knoten

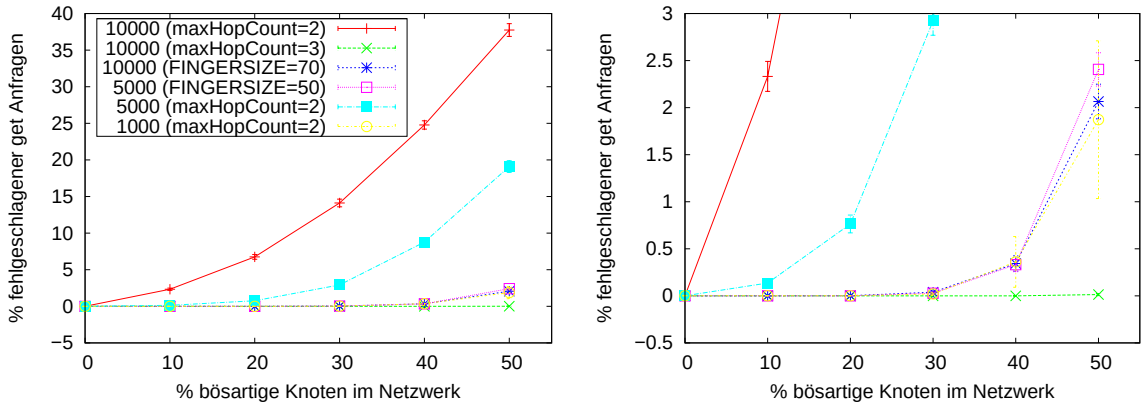


Abbildung 5.27: Fehlgeschlagene *get* Anfragen bei unterschiedlichen Netzwerkgrößen

unterschiedlichen Anteilen bösartiger Peers vorgenommen. Alle Messungen finden in einem dynamischen Netzwerk statt, das sich durchschnittlich alle fünf Simulationsschritte ändert. Der Simulationsaufbau entspricht größtenteils dem Aufbau für die dynamischen Messungen im vorhergehenden Kapitel 5.2.2. Anstatt die Anzahl der Gruppen zu variieren, wird hier der Anteil bösartiger Knoten verändert. Bei den dargestellten Ergebnissen handelt es sich um die Durchschnitte von 10000 *put* und *get* Anfragen, wobei Knoten und Datenobjekte zufällig ausgewählt werden. Die Simulationen wird hierbei zehnmal mit unterschiedlicher „Seed“ des Zufallszahlengenerators wiederholt und deren Durchschnitt berechnet. Die *put* und *get* Anfragen verteilen sich auf 20000 Simulationsschritte. So ergeben sich insgesamt 4000 Änderungen im Netzwerk.

Um ein verlässliches P2P-Netzwerk zu gewährleisten, werden dessen Einstellungen entsprechend gewählt (insbesondere die Anzahl der Fingereinträge $FINGERSIZE$). Die Standardeinstellung ist hierbei wie zuvor 32. Sobald bösartige Knoten im Netzwerk vorhanden sind, die beim Flooding Nachrichten nicht mehr weiterleiten, ist die Erreichbarkeit der Daten für größere Netzwerke, wie in Abbildung 5.27 ersichtlich, trotz Datenreplikation nicht mehr garantiert. Bei 10000 Knoten und $maxHopCount = 2$ ist schon bei 10% bösartiger Knoten im Netzwerk die Fehlerrate größer als 2%. Bei 5000 Knoten im Netzwerk ist dies erst bei 30% bösartiger Knoten der Fall. Da eine Erhöhung von $maxHopCount$ von zwei auf drei unverhältnismäßig erscheint,[1] wird die Anzahl an Fingereinträgen erhöht, bis auch bei 30% bösartiger Knoten die Daten zuverlässig gefunden werden. Dies bedingt einen höheren Verwaltungsaufwand für die Fingertabelleneinträge. Aufgrund der, gegenüber der Erhöhung von $maxHopCount$, geringeren Anzahl an Nachrichten bei Datenanfragen ist dies vertretbar. Für 5000 Knoten wird $FINGERSIZE = 50$ und für 10000 Knoten $FINGERSIZE = 70$ gewählt. Mit diesen Einstellungen kann, wie in Abbildung 5.27 ersichtlich, bei bis zu 30% bösartiger Knoten die Fehlerrate bei 0% gehalten werden. Zum Vergleich wird hier ebenfalls noch die Messung mit $maxHopCount = 3$ und $FINGERSIZE = 32$ für 10000 Knoten aufgeführt. Mit diesen Einstellungen ist die Fehlerrate selbst bei 50% bösartiger Knoten bei 0%.

Die Anzahl der Nachrichten, insbesondere der *get* Request-Nachrichten, die durch den Flooding Algorithmus bei den verschiedenen vorgestellten Einstellungen erzeugt werden, ist

[1] Die maximale Anzahl der Nachrichten berechnet sich aus $FINGERSIZE^{maxHopCount}$

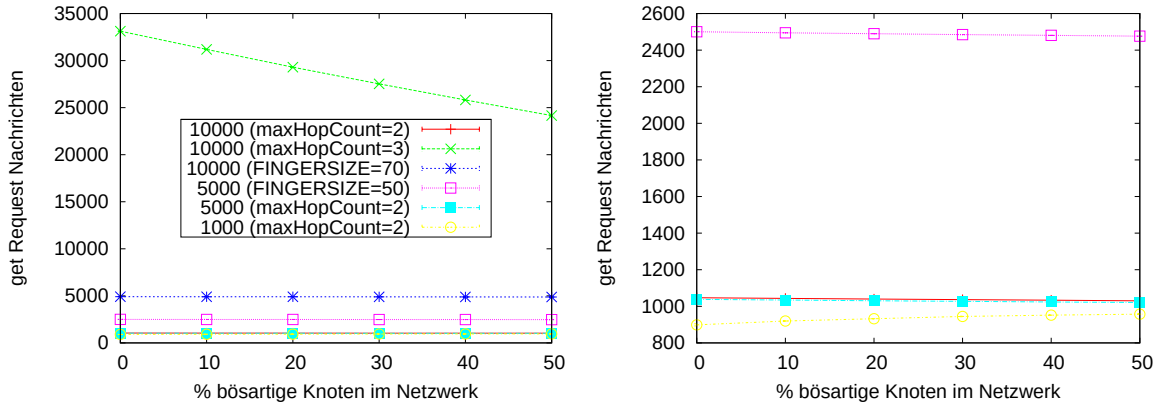


Abbildung 5.28: Anzahl *get* Request-Nachrichten bei unterschiedlichen Netzwerkgrößen

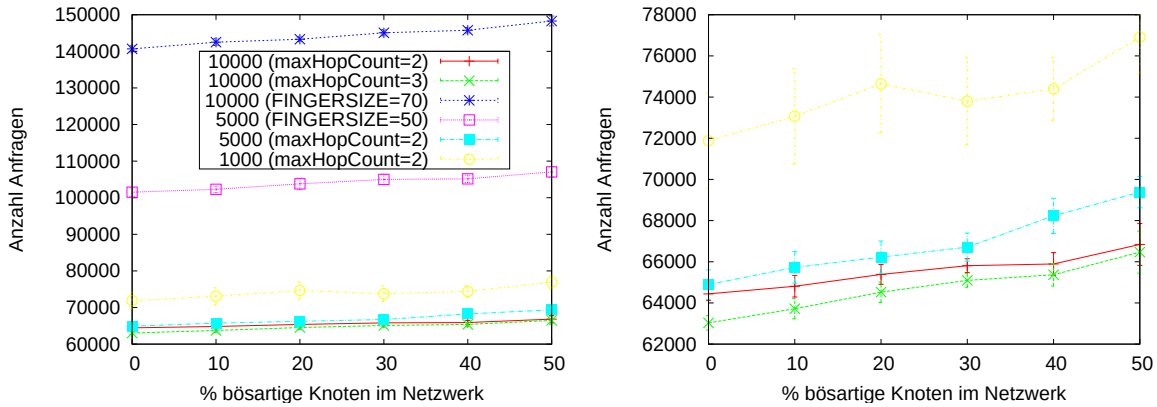
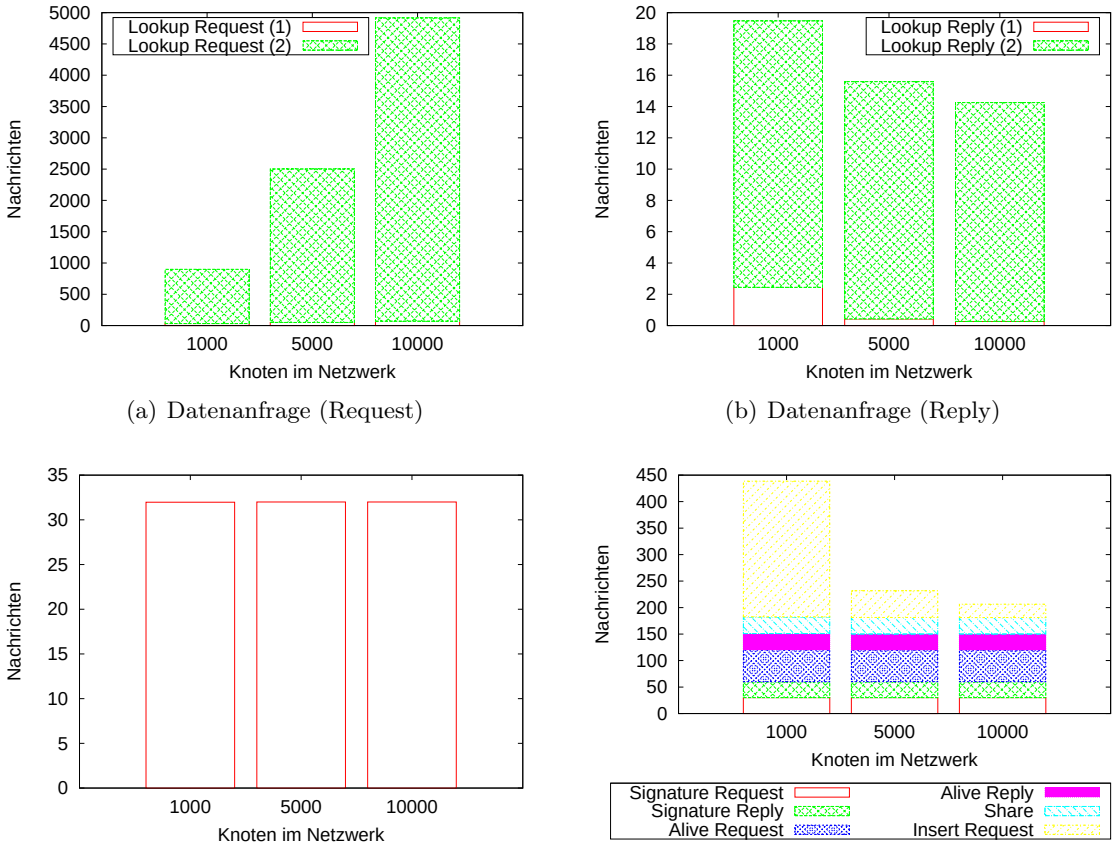


Abbildung 5.29: Anzahl FingerEntry-Nachrichten zur Suche neuer Fingereinträge bei verschiedenen Netzwerkgrößen und Einstellungen

in Abbildung 5.28 ersichtlich. Die Unverhältnismäßigkeit der Erhöhung des $maxHopCount$ auf drei wird hier besonders deutlich. Bei $maxHopCount = 3$ und 10000 Knoten ist zu sehen, dass die Anzahl der erzeugten *get* Request-Nachrichten mit Zunahme der bösartigen Peers abnimmt, da diese die *get* Request-Nachrichten nicht mehr an andere Peers weiterleiten. Gut zu sehen ist auch wie sich die Anzahl der erzeugten Nachrichten für $maxHopCount = 2$ und $FINGERSIZE = 32$ zwischen den Netzwerkgrößen 5000 und 10000 kaum unterscheidet, während bei 1000 Knoten deutlich weniger Nachrichten erzeugt werden. Dies ist dem Einsparungseffekt der History bei den Anfragen zu verdanken. Diese verhindert, dass eine Nachricht zweimal zum selben Knoten gesendet wird. Hierdurch kommt es zu der hier beobachteten Nachrichtenanzahl unterhalb des theoretischen Wertes von 32^2. Zuletzt ist noch zu erwähnen, dass durch die Erhöhung von $FINGERSIZE$ auf 50 bzw. 70 die Anzahl der Nachrichten deutlich steigt. Dies ist gegenüber der Erhöhung des $maxHopCount$ jedoch das deutlich effizientere Vorgehen.

Durch die Erhöhung der Anzahl der Fingereinträge erhöht sich der Aufwand zur Überprüfung dieser Einträge. Dies ist in Abbildung 5.29 zu sehen. Hier wird die Anzahl FingerEntry-Nachrichten über die gesamte Simulationszeit dargestellt. Diese Nachrichten werden versendet, wenn ein Peer feststellt, dass einer seiner Fingereinträge sich aus dem Netzwerk verab-



(a) Datenanfrage (Request)

(b) Datenanfrage (Reply)

(c) Speichern von Datenobjekten

(d) Änderung einer Replikationsgruppe

Abbildung 5.30: Aufwand der Grundoperationen im unstrukturierten P2P-Netzwerk

schiedet hat und er für diesen einen Ersatz ermitteln möchte. Aufgrund der größeren Fingertabellen bei $FINGERSIZE$ 50 bzw. 70 ergeben sich Änderungen an den Fingertabellen erheblich häufiger. Zudem ist die Anzahl Anfragen umso höher, desto kleiner das Netzwerk ist. Dies liegt im Simulationsaufbau begründet. Da jeweils 4000 Änderungen am Netzwerk vorgenommen werden, sind die Knoten umso häufiger von Änderungen betroffen, je kleiner das Netzwerk ist. Entsprechend ist die Anzahl der FingerEntry-Nachrichten in diesen Netzwerken höher.

Alle anderen Teile des Protokolls mit den entsprechenden Nachrichten sind von der Netzwerkgröße und dem Anteil bösartiger Peers unabhängig. Das Hinzukommen eines Knotens zum Netzwerk verursacht stets einen Aufwand von $FINGERSIZE * NUMBOOTSTRAP$ Nachrichten. Der Aufwand von Gruppenänderungen bleibt ebenfalls konstant (siehe Abbildung 5.26).

5.2.4 Speichern und Abfragen von Datenobjekten

Das unstrukturierte P2P-Netzwerk mit aktiver Replikation, wie es in PACS realisiert ist, speichert Daten, beantwortet Datenabfragen und verwaltet die Replikate bei Veränderungen der Replikationsgruppen. Der Aufwand für diese Aufgaben ist in Abbildung 5.30 dargestellt.

Anzahl Knoten	**1000**	**5000**	**10000**
$maxHopCount$	2	2	2
$FINGERSIZE$	32	50	70
$NUMGROUPS$	30	150	300

Tabelle 5.1: Einstellungen PACS unstrukturierter P2P-Netzwerkspeicher

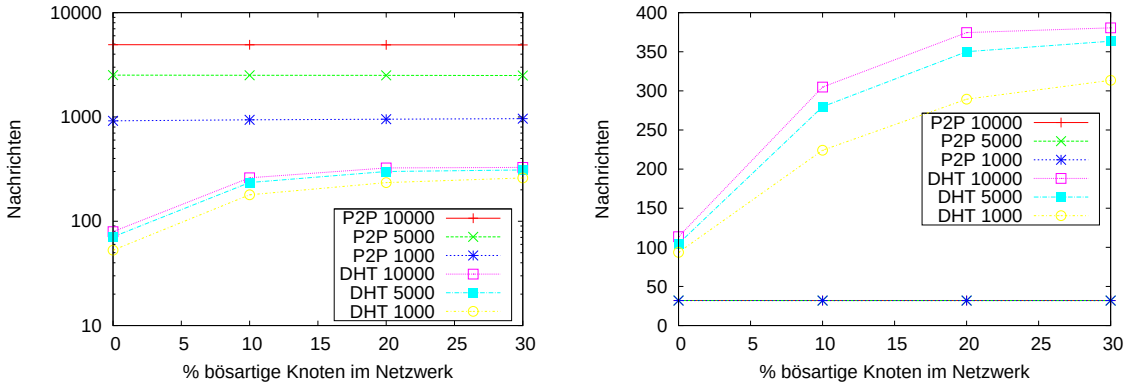
Es handelt sich hierbei um drei dynamische Netzwerke mit einer Änderung des Netzwerk alle fünf Simulationsschritte mit 1000, 5000 und 10000 Knoten. Da in einem Netzwerk ohne bösartige Knoten die meisten Nachrichten erzeugt werden, wird für die Gegenüberstellung jeweils angenommen, dass keine bösartigen Peers im Netzwerk vorhanden sind. Wie aus den Messungen zuvor (siehe Abbildung 5.28) hervorging, hat der Anteil bösartiger Knoten im Netzwerk ohnehin kaum einen Einfluss auf den Aufwand dieser Operationen. Die Ergebnisse für 10% bis 50% bösartige Peers im Netzwerk sind im Anhang F dargestellt. Die übrigen Einstellungen für die Netzwerke sind in Tabelle 5.1 aufgeführt und unterscheiden sich nicht von denen der vorhergehenden Messungen. Selbiges gilt für den Simulationsaufbau. In Abbildung 5.30(a) wird jeweilig der Aufwand für eine Datenabfrage innerhalb der drei Netzwerke einander gegenübergestellt. Er nimmt mit zunehmender Netzwerkgröße zu. Die Antworten der Peers für diese Anfragen sind in Abbildung 5.30(b) dargestellt. Während Request-Nachrichten nur eine geringere Datengröße aufweisen, werden bei den hier dargestellten Reply-Nachrichten die gespeicherten Datenobjekte übertragen. Der Gesamtaufwand einer Datenanfrage ist die Summe aus Request- und Reply-Nachrichten. Die Lookup-Request- und Reply-Nachrichten werden hierbei noch nach ihrem $HopCount$ kategorisiert. Aufgrund des exponentiellen Wachstums ist beim Hauptteil der Nachrichten $HopCount = 2$. Die Abnahme der Anzahl der Request- sowie Reply-Nachrichten wird verursacht durch die Einstellungen, die für $maxHopCount$ bzw. $FINGERSIZE$ für die jeweiligen Netzwerkgrößen gewählt wurden. Entscheidend ist hier, dass der Aufwand einer *get* Anfrage mit zunehmender Netzwerkgröße linear steigt.

Der Aufwand der Datenspeicherung entspricht der Größe der Replikationsgruppe, da an jedes Mitglied dieser Gruppe das neue Datenobjekt übermittelt werden muss. Der Aufwand bleibt deshalb, wie in Abbildung 5.30(c) dargelegt, konstant bei 32 Nachrichten.

Abbildung 5.30(d) zeigt schließlich den Aufwand, den eine Änderung der Replikationsgruppe erzeugt. Neben der Verwaltung der Gruppe (Signatur-, Alive- und Share-Nachrichten) sind hier die Insert-Nachrichten zu erwähnen, die die in der Replikationsgruppe gespeicherten Datenobjekte übertragen. Bei den bei diesen Simulationen gewählten Einstellungen unterscheidet sich die Anzahl der Datenobjekte pro Replikationsgruppe erheblich, denn es werden immer 10000 Datenobjekte gespeichert, während die Anzahl der Replikationsgruppen für die drei Netzwerke variiert.

5.2.5 Zusammenfassung der Ergebnisse

Anhand der Messungen wird die Funktionstüchtigkeit des unstrukturierten Netzwerks manifestiert. Deutlich wird allerdings auch der Aufwand, der damit verbunden ist, diese bei bösartigen Knoten im Netzwerk zu gewährleisten. Die Erzeugung von bis zu 5000 Nachrichten bei einer Anfrage eines Datenobjekts (obwohl keine bösartigen Knoten im Netzwerk vorhanden sind) stellt einen großen zusätzlichen Aufwand dar. Allerdings erzeugt der Flooding Algorithmus selbst ohne bösartige Peers bei 10000 Knoten ca. 1000 Nachrichten um ein Datenobjekt abzufragen. Die Ursache des Aufwands des unstrukturierten P2P-Netzwerks liegt deshalb in



(a) Datenanfrage

(b) Speichern von Datenobjekten

Abbildung 5.31: Vergleich DHT- und P2P-Netzwerk

der Funktionsweise des Flooding Algorithmus. Effizientere Verfahren, die in Kapitel 2.7.1 vorgestellt werden, sind aber wesentlich anfälliger für Manipulationen durch bösartige Peers und sind deshalb für die hier betrachteten Umgebungen nicht geeignet.

Die Veränderung der Replikationsgruppe stellt hingegen keinen nennenswerten Aufwand dar. Eine solche Änderung beschränkt sich auf die Knoten der Replikationsgruppe und den Gruppeneigentümer und erzeugt deshalb maximal 175 Nachrichten plus die für die Datenobjektübertragung nötigen Insert-Nachrichten.

5.3 Vergleich DHT und unstrukturiertes P2P-Netzwerk

Bei der Gegenüberstellung der beiden Speichertechniken DHT und unstrukturiertes Netzwerk, treten die Unterschiede klar hervor. Die vergleichende Abbildung 5.31 veranschaulicht dies. „P2P“ bezeichnet hierbei das unstrukturierte Netzwerk mit definierter Replikationsgruppe und „DHT“ die verlässliche DHT-Anwendung, die beide für PACS entwickelt wurden. Bei den hier dargestellten Messungen handelt es sich um dynamische Messungen, wobei sich das Netzwerk alle fünf Simulationsschritte verändert. Die Einstellungen für die DHT und das unstrukturierte Netzwerk entsprechen jenen, die in den vorhergehenden Kapiteln für ein zuverlässiges Arbeiten dieser Protokolle vorgestellt wurden. Wird die Anzahl der Nachrichten für eine Datenanfrage verglichen, wie in Abbildung 5.31(a) zu sehen, tritt der Vorteil der verlässlichen DHT klar zu Tage. Der Aufwand zum Abfragen eines Datenobjekts ist mit weniger als 500 Nachrichten wesentlich geringer, als die minimal 1000 und maximal 5000 Nachrichten beim unstrukturierten Netzwerk. Zudem kommt dem strukturierten Netzwerk seine bessere Skalierbarkeit bei größeren Netzwerken zu gute. Hier liegt die maximale Zunahme an Nachrichten von 1000 zu 10000 Knoten bei 90 Nachrichten, während es beim unstrukturierten P2P-Netzwerk 4000 Nachrichten sind.

Beim Speichern von Datenobjekten, welches in Abbildung 5.31(b) dargestellt ist, schneidet die DHT hingegen schlechter ab als das unstrukturierte Netzwerk. Grund ist hier die Suche nach dem für das Speichern zuständigen Peer. Dies entfällt beim unstrukturierten Netzwerk, bei dem das Datenobjekt lediglich lokal gespeichert und auf die Knoten der Replikationsgruppe übertragen wird. Deshalb erfordert das Speichern der Datenobjekte im unstrukturierten Netzwerk unabhängig von der Netzwerkgröße ca. 32 Nachrichten.

Änderungen im Netzwerk verursachen bei beiden Arten von Netzwerken zusätzlichen Aufwand in der Netzwerkverwaltung. Beim unstrukturierten Netzwerk wird dies durch die mögliche Veränderung der Replikationsgruppe verursacht, während bei der DHT vor allem die korrekte Einordnung im Netzwerk und das Füllen der Fingertabelle von neuen Knoten ursächlich ist. Beide Vorgänge lassen sich von dem durch sie erzeugten Aufwand her, gut miteinander vergleichen, wenngleich sie von verschiedenen Faktoren abhängen, wie Anzahl der gespeicherten Datenobjekte (unstrukturiert) und Netzwerkgröße (DHT). Auch der Verwaltungsaufwand der Netzwerke ist vergleichbar. Beide müssen regelmäßig überprüfen, ob sich die Knoten ihrer Fingertabelle noch im Netzwerk befinden. Diese Überprüfung ist allerdings für die DHT wesentlich wichtiger, als für das unstrukturierte Netzwerk, und läuft bei der DHT deshalb häufiger ab. Zudem müssen beide Netzwerkarten für die Replikation der gespeicherten Datenobjekte sorgen. Die Verwaltung und Überprüfung der Replikationsgruppen ist ebenfalls bei beiden Netzwerkarten nötig, wenngleich diese beim unstrukturierten Netzwerk aufwändiger ist. Hervorzuheben ist noch, dass die verlässliche Speicherung der Daten im unstrukturierten Netzwerk mit aktiver Replikation im Gegensatz zur DHT selbst bei Anteilen von bösartigen Knoten über 30% zuverlässig gewährt werden kann.

Für den hier dargestellten Anwendungsfall sind Datenabfragen die Hauptoperationen. Deshalb wird der fast zwölffache Aufwand der DHT gegenüber dem unstrukturierten P2P-Netzwerke beim Speichern der Daten durch den bis zu Faktor 68 niedrigeren Aufwand bei Datenabfragen mehr als ausgeglichen. Hier gibt die bessere Skalierbarkeit der DHT bei großen Netzwerken den Ausschlag. Hinzu kommt die effektive Eingrenzung der Kosten der DHT durch den Fehlertest, falls keine bösartigen Knoten im Netzwerk vorliegen. Die verlässliche Speicherung der Daten im unstrukturierten Netzwerk ist deshalb nur für sehr kleine und dynamische Netzwerke geeignet. Für größere und eher statische Netzwerke ist die verlässliche DHT aufgrund ihrer höheren Effizienz vorzuziehen.

5.4 Komplexitätsanalyse der verteilten Durchsetzung der Zugriffskontrolle

Die für PACS entwickelte clientseitige Durchsetzung der Privilegien wird in Kapitel 3.8 erläutert und in [SHS09] inklusive der Komplexitätsabschätzung publiziert. Indem sie Elemente aus gruppenbasierter und clientseitiger Durchsetzung kombiniert, stellt sie einen Kompromiss zwischen diesen beiden Durchsetzungsvarianten dar. Die Auswirkung dieser Kombination auf die Leistungsfähigkeit und den Aufwand der Durchsetzung soll an dieser Stelle genauer untersucht werden. Die für PACS entwickelte Durchsetzungsvariante wird hierbei mit der ausschließlich clientseitigen Durchsetzung (siehe Kapitel 2.5.2), im Folgenden „Client“ abgekürzt, und der ausschließlich gruppenbasierten Durchsetzung (siehe Kapitel 2.5.3), im Folgenden „Gruppe“ genannt, verglichen.

Um die Übersichtlichkeit des Vergleichs zu erhöhen, werden jeweils nur die einfachsten Formen aller Durchsetzungsvarianten verglichen. Entsprechend werden auch die beschriebenen Optimierungen der Durchsetzungsvarianten von PACS beim Vergleich nicht berücksichtigt. Für diesen wird lediglich die Anzahl an Nachrichten, die übertragen werden, und die zu übertragende Datenmenge betrachtet. Um die Untersuchung weiter zu vereinfachen wird angenommen, dass die Daten in einer DHT gespeichert sind und mit einen Aufwand von $\mathcal{O}(\log N)$ Nachrichten abgefragt werden können, wobei N hierbei der Größe des Netzwerks entspricht.

	get (cold)	*get* (hot)	*put*	- Priv.	+ Priv.
Client	$2 * log(N)$	$log(N)$	$2 * log(N)$	$4 * log(N)$	$log(N)$
Gruppe	$(t + 1) * log(N) + 2t$	$t * log(N) + 2t$	$2 * log(N)$	0	0
PACS	$2 * log(N) + 2t$	$log(N)$	$2 * log(N)$	$4 * log(N) + 2t$	0

Tabelle 5.2: Anzahl der Nachrichten der Durchsetzungsvarianten

	get (cold)	*get* (hot)	*put*	- Priv.	+ Priv.
Client	$SD + o$	o	$SD + o$	$2SD + 2o$	SD
Gruppe	$DGIO + 2t * o$	$2t * o$	$DGIO + o$	0	0
PACS	$DGIO + t * sig + o$	o	$DGIO + o$	$2*(DGIO+t* sig + o)$	0

Tabelle 5.3: Zu übertragende Datenmenge der Durchsetzungsvarianten

In der Tabelle 5.2 steht t für die Anzahl an Stellvertretern die für eine erfolgreiche Ver- und Entschlüsselung kontaktiert werden müssen, o entspricht dem Datenobjekt und sig einer digitalen Signatur. Zudem wird bei der clientseitigen Durchsetzung angenommen, dass die für das Entschlüsseln nötigen Schlüssel in einer Schlüsseldatei in der DHT abgelegt sind. Die Größe dieser Schlüsseldatei (SD) soll hierbei zur besseren Vergleichbarkeit jener des DGIO entsprechen. Der vertrauliche Austausch der Daten wird bei der Gegenüberstellung der Durchsetzungsarten nur berücksichtigt, wenn sich zwischen den Varianten hierbei Unterschiede ergeben. Beim Rechenaufwand wird zwischen Entschlüsselung (dec), Verschlüsselung (enc) und der Erzeugung digitaler Signaturen ($sign$) unterschieden. Zu beachten ist weiterhin, dass der Aufwand für die Privilegienverwaltung nicht Bestandteil dieses Vergleichs ist. Analysiert werden der Aufwand für die Durchsetzung der Datenoperationen *put* und *get*, das Aktualisieren eines Datenobjekts sowie der Aufwand für das Entziehen und Gewähren eines Privilegs. Bei den Datenoperationen wird zudem zwischen der ersten Anfrage („cold“) und den folgenden Anfragen („hot“) unterschieden, um das lokale Zwischenspeichern der Schlüssel bei den Clients zu berücksichtigen. Die Ergebnisse dieser Abschätzung sind in den Tabellen 5.2, 5.3 und 5.4 zu sehen (siehe auch Sturm u.a. [SHS09]).

***get* (cold)** Hierbei handelt es sich um Datenanfragen, bei denen der Schlüssel noch nicht lokal zwischengespeichert ist. Entsprechend bedarf es bei der clientseitigen Durchsetzung einer DHT-Anfrage für die Schlüsseldatei. Der Rechenaufwand besteht aus dem Entschlüsseln der Schlüsseldatei und des angefragten Datenobjekts o. Beim von PACS eingesetzten Durch-

	get (cold)	*get* (hot)	*put*	- Priv.	+ Priv.
Client	$2 * dec$	dec	enc	$2 * dec + enc$	enc
Gruppe	$t * enc + 2t * dec$	$t * enc + 2t * dec$	enc	0	0
PACS	$t*(sign+enc+dec)+ dec$	dec	enc	$2t * (sign + enc + dec) + enc + dec$	0

Tabelle 5.4: Rechenaufwand der Durchsetzungsvarianten

setzungsverfahren muss das DGIO abgefragt werden. Anschließend müssen t Stellvertreter angefragt werden und eine verschlüsselte Teilsignatur zurückliefern. Die Abfrage des Datenobjekts resultiert in einer weiteren DHT-Anfrage. Bei der gruppenbasierten Durchsetzung wird ebenfalls ein DGIO abgefragt, um die für das Objekt zuständigen Stellvertreter zu ermitteln. Anschließend werden t Stellvertreter mit der Teilentschlüsselung des Datenobjekts o und seiner sicheren Übermittlung an den Anfragenden beauftragt.

***get* (hot)** Wurde ein Datenobjekt schon einmal von Client angefragt und der für die Entschlüsselung des Datenobjekts hierbei erforderliche Schlüssel lokal zwischengespeichert, kann dieser bei einer wiederholten Datenanfrage erneut für die Entschlüsselung benutzt werden und muss nicht neu abgefragt werden. Voraussetzung dafür ist, dass der Schlüssel für dieses Datenobjekt immer noch gültig ist. Ein Schlüssel wird bei den Durchsetzungsvarianten Client und PACS durch das Entziehen eines Privilegs für das Datenobjekt ungültig.

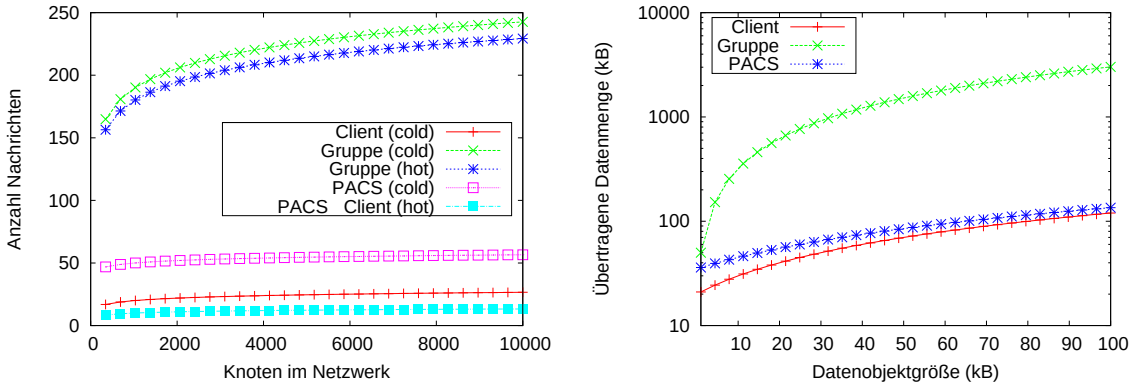
put Das Einfügen eines neuen Datenobjekts führt zu einer Modifikation des DGIO bzw. der Schlüsseldatei. Diese müssen entsprechend im Netzwerk gespeichert werden. Zudem muss das verschlüsselte Datenobjekt in der DHT gespeichert werden.

Aktualisieren eines gespeicherten Datenobjekts Bei der Aktualisierung eines gespeicherten Datenobjekts muss dieses bei allen drei Durchsetzungsvarianten zunächst verschlüsselt und anschließend im Netzwerk gespeichert werden. Entsprechend ist eine Verschlüsselungsoperation *enc* erforderlich. Beim Speichern wird das Datenobjekt o übertragen sowie $\log(N)$ Nachrichten erzeugt. Da sich die drei Ansätze hierbei nicht unterscheiden wird diese Datenoperation nicht weiter betrachtet.

- Priv. Wird ein zuvor gewährtes Privileg an einem Objekt o widerrufen, muss dieses Objekt beim Client und PACS-Ansatz mit einem neuen Schlüssel verschlüsselt werden. Entsprechend ist das Abfragen, das Entschlüsseln, das erneute Verschlüsseln und schließlich das Speichern des Objekts erforderlich. In PACS muss zudem von den t Stellvertretern ein neuer Schlüssel für die Verschlüsselung angefordert werden. Dies resultiert in zusätzlichen $2t$ Nachrichten. Bei der gruppenbasierten Verschlüsselung entsteht kein Aufwand, da der Schlüssel zur Entschlüsselung des Datenobjekts keinem Teilnehmer bekannt ist und deshalb auch nicht erneuert werden muss.

+ Priv. Hierbei handelt es sich um die Aktion zum Gewähren eines neuen Privilegs. Bei der clientseitigen Durchsetzung muss hierzu die Schlüsseldatei angepasst und gespeichert werden. Alle anderen Ansätze verursachen hierbei keinen Aufwand.

In Abbildung 5.32 wird die prognostizierte Anzahl an Nachrichten und Datenmengen für die verschiedenen Ansätze für Netzwerke mit bis zu 10000 Knoten graphisch dargestellt. Wie zu sehen, ist der PACS-Ansatz bei *get* Anfragen der gruppenbasierten Durchsetzung deutlich überlegen. Wenn die Schlüssel lokal zwischengespeichert vorliegen, verhält er sich gleich dem Ansatz mit clientseitiger Durchsetzung. Bei der zu übertragenden Datenmenge ist klar ersichtlich, dass das Transfervolumen bei PACS wesentlich niedriger ist als bei der gruppenbasierten Durchsetzung – es ist sogar mit dem Transfervolumen der clientseitigen Durchsetzung vergleichbar.



(a) Anzahl der Nachrichten für *get* Anfragen mit $t = 15$ and $log_2(n)$

(b) Übertragene Datenmenge für *get* (cold) Anfragen ($sig = 1$ kB, DGIO $= 20$ kB, $t = 15$, $log_2(n)$)

Abbildung 5.32: Vergleich der verschiedenen Durchsetzungsarten

Resultat dieser Komplexitätsanalyse ist, dass PACS einen gelungenen Kompromiss zwischen clientseitiger und gruppenbasierter Durchsetzung darstellt. PACS unterstützt administrative Delegation und Datenreplikation, ohne dabei einen wesentlich höheren Aufwand zu erzeugen. Des Weiteren wird deutlich, dass der Aufwand der gruppenbasierten Durchsetzung insbesondere für größere Netzwerke und Datenobjekte nicht mehr vertretbar ist.

5.5 Vergleich der Varianten des Privilegienspeichers

Der Vergleich zwischen dem DHT-basierten Privilegienspeicher (DHT) und dem Privilegienspeicher basierend auf einem unstrukturierten P2P-Netzwerk mit aktiver Replikation (P2P) beim Einsatz der clientseitigen und serverseitigen Durchsetzungsvarianten zeigt dessen Unterschiede deutlich auf. Dies wird in den vergleichenden Abbildungen besonders deutlich.

Grundlage sind hierfür die in den Kapiteln G.1 und G.2 im Anhang vorgestellten Aufwandsberechnungen. Die in den Berechnungen enthaltenen *get*, *put*, *get_data* und *put_data* Aufrufen werden für den hier dargestellten Vergleich durch die für diese Aufrufe gemessenen Werte für ein dynamisches Netzwerk mit einer Änderung im Netzwerk alle fünf Simulationsschritte (wie in Kapitel 5.3 vorgestellt) ersetzt. Da die übertragene Datenmenge von vielen weiteren Faktoren wie Größe der Export Policy u.a. abhängt, wird lediglich die Anzahl der Nachrichten dargestellt. Zunächst werden die beiden Privilegienspeicher einander bei der serverseitigen Durchsetzung gegenübergestellt.

5.5.1 Serverseitige Durchsetzung

Aufgrund der unterschiedlichen Organisation der P2P-Netzwerke, welche die zwei Arten des Privilegienspeichers benutzen, unterscheidet diese Netzwerke vor allem der Aufwand der Operationen *get*, *put* und *delete*, wobei *delete* bei den folgenden Ausführungen dem Aufwand von *put* gleichgesetzt wird. Daneben unterscheiden sich beide Varianten geringfügig im Ablauf der Durchsetzung und Verwaltung der Privilegien. Die Unterschiede zwischen der verlässlichen DHT und dem unstrukturierten P2P-Netzwerk mit aktiver Replikation wird bereits in

Kapitel 5.3 erläutert. Die dort getätigten Aussagen gelten auch für den Vergleich der Privilegienspeicher.

Die beiden Privilegienspeicher unterscheiden sich vor allem beim Gewähren und Widerrufen von Benutzerzuordnungen, sowie beim Erteilen administrativer Privilegien an Rollen. Zur Veranschaulichung wird der Aufwand in Anzahl an Nachrichten für die drei Aktionen in Abbildung 5.33 dargestellt.

Aus Abbildung 5.33(a) geht hervor, dass beim Hinzufügen von Benutzerzuordnungen der Aufwand des unstrukturierten P2P-Privilegienspeichers geringer ist. Während der maximale Aufwand beim unstrukturierten P2P-Privilegienspeicher hier bei ca. 5000 Nachrichten liegt, kann er beim DHT-Privilegienspeicher bis zu 11000 Nachrichten betragen. Auch die minimale Schranke liegt beim unstrukturierten P2P-Speicher mit ca. 32 Nachrichten wesentlich niedriger, als beim DHT-Speicher mit 300 bis 400 Nachrichten. Dies resultiert vor allem aus der aufwändigeren Verwaltung der Benutzerzuordnungen beim DHT Privilegienspeicher.

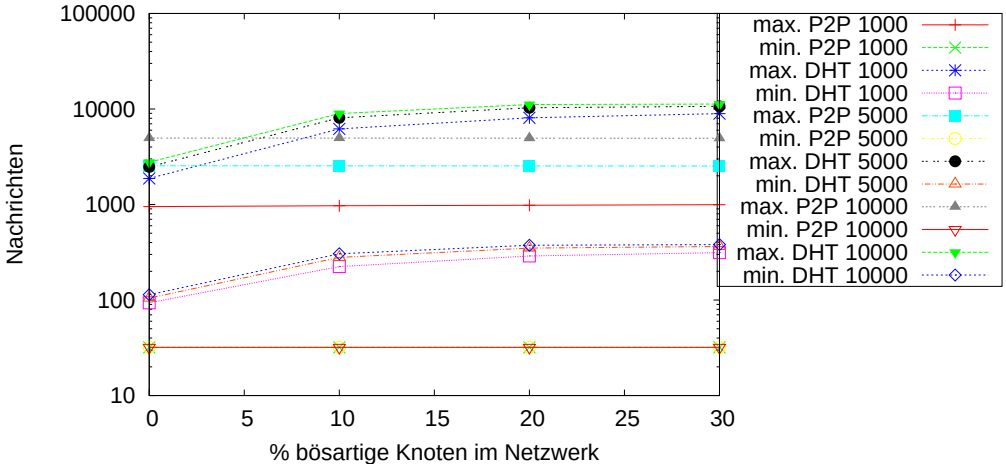
Ist die Organisation nahezu identisch, wie beim in Abbildung 5.33(b) gezeigten Erteilen von administrativen Privilegien an Rollen, ist die DHT wesentlich effizienter. Bei 10000 Knoten beträgt der maximale Aufwand des unstrukturierten P2P-Speichers hier ca. 10000 Nachrichten. Im Vergleich zum DHT-Speicher mit ca. 700 Nachrichten ist dies ein signifikanter Unterschied.

Beim Widerruf von Benutzerzuordnungen in Abbildung 5.33(c) wird deutlich, dass der DHT-Privilegienspeicher im optimalen Fall durchaus effizienter sein kann, als der unstrukturierte P2P-Privilegienspeicher (375 zu 5000 Nachrichten bei 10000 Knoten), allerdings im schlechtesten Fall auch wesentlich schlechter (11000 zu 5000 Nachrichten bei 10000 Knoten).

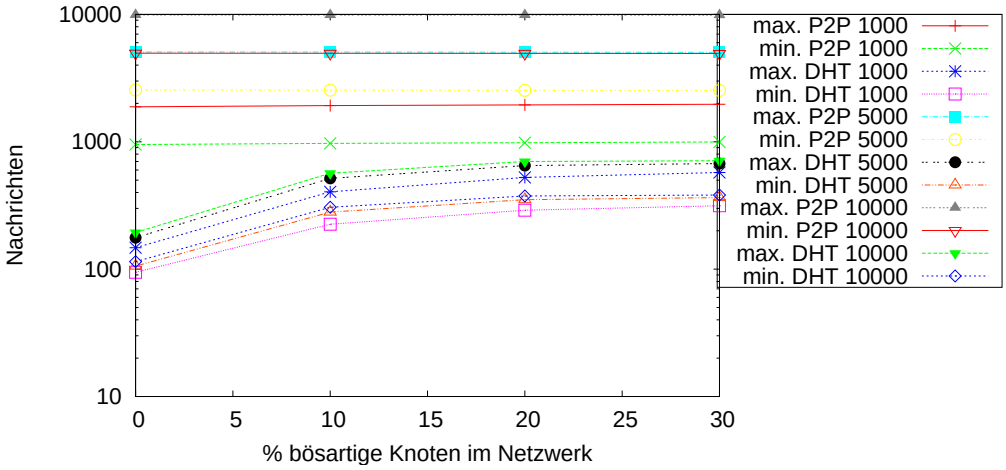
Die häufigste Aktion eines Privilegienspeichers ist die Durchsetzung von Privilegien bei Datenanfragen. Der Vergleich der beiden Speicherarten in Abbildung 5.34 zeigt, dass hierbei der Aufwand des DHT-Privilegienspeichers wesentlich kleiner ist. Es handelt sich hier in beiden Fällen maximal um einen *get* Aufruf. Dieser ist jedoch bei der DHT, wie schon zuvor in Kapitel 5.3 dargelegt, wesentlich effizienter. Insbesondere wächst der Abstand zwischen den beiden Privilegienspeichern mit zunehmender Netzwerkgröße. So ist die DHT im Extremfall bei 10000 Knoten und 0% bösartiger Knoten um den Faktor 62 effizienter.

Insgesamt ist der DHT-Speicher dem unstrukturierten P2P-Privilegienspeicher bei der serverseitigen Durchsetzung aufgrund der Messergebnisse und Erläuterungen klar vorzuziehen. Zum einen liegt dies an der erhöhten Effizienz des DHT-Speichers, der die Vorteile des unstrukturierten P2P-Speichers bezüglich der Privilegienverwaltung und der Speicherung von Privilegien mehr als aufhebt. Zum anderen besitzt die DHT durch den Fehlertest die Eigenschaft, ihren Aufwand dem tatsächlich vorherrschenden Bedrohungsszenarios im Netzwerk anzupassen. Der unstrukturierte P2P-Speicher hingegen hat diese Eigenschaft nicht und arbeitet deshalb immer mit dem Aufwand des maximalen Bedrohungsszenario. Deshalb ist der Abstand zwischen den beiden Privilegienspeichern auch bei Messergebnissen mit 0% bösartigen Peers im Netzwerk am größten. Da die DHT eine bessere Skalierbarkeit besitzt als der unstrukturierte P2P-Speicher, ist der Vorteil der DHT umso größer, je größer das Netzwerk ist. Allerdings ist die für PACS entwickelte verlässliche DHT für sehr dynamische Netzwerke nicht geeignet.

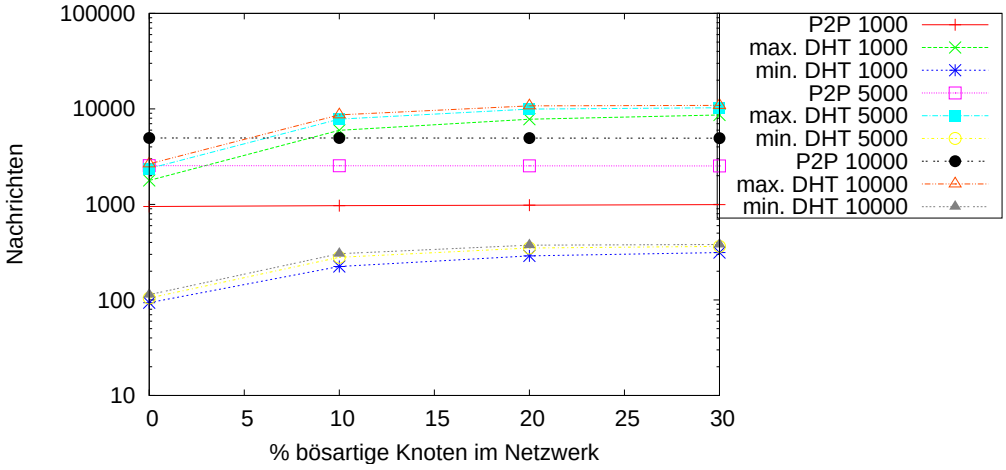
Hier zeigt sich der Vorteil des unstrukturierten P2P-Privilegienspeichers, der eine größere Robustheit gegenüber Änderungen im Netzwerk besitzt. Für kleine, sehr dynamische Umgebungen kann er deshalb durchaus eine Alternative darstellen.



(a) Hinzufügen einer Benutzerzuordnung



(b) Hinzufügen eines administrativen Privilegs zu einer Rolle



(c) Widerrufen einer Benutzerzuordnung

Abbildung 5.33: Aufwand der Verwaltung von Privilegien

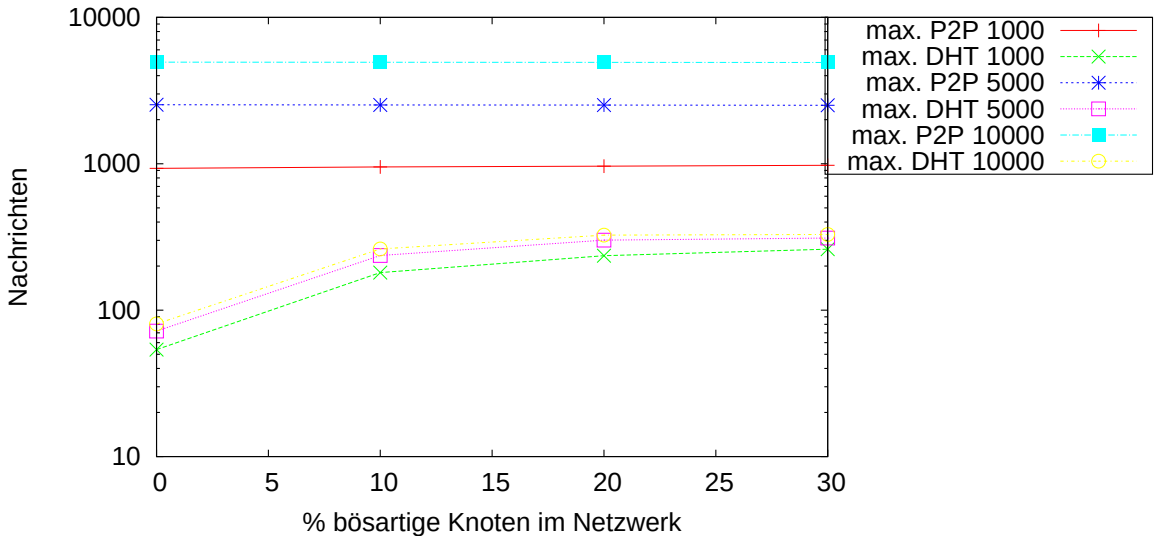


Abbildung 5.34: Aufwand der serverseitigen Durchsetzung bei Datenanfragen

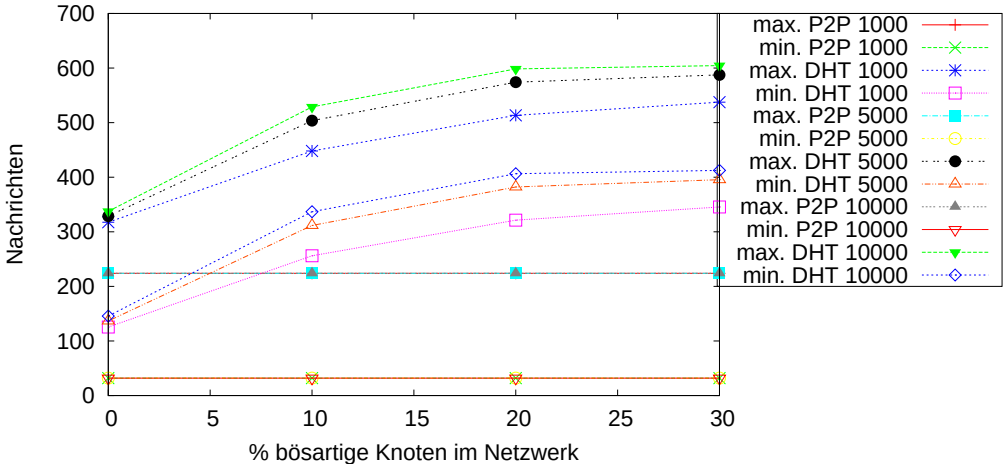
5.5.2 Clientseitige Durchsetzung

Aufgrund der weitgehend identischen Organisation der Privilegien in den beiden Varianten der Privilegienspeicher unterscheidet sich der Aufwand der clientseitigen Durchsetzung vor allem im Aufwand der Operationen *get*, *put*, *delete*, *get_data*, *put_data* und *delete_data*, wobei *delete* wie bei der serverseitigen Durchsetzung dem Aufwand von *put* gleichgesetzt wird.

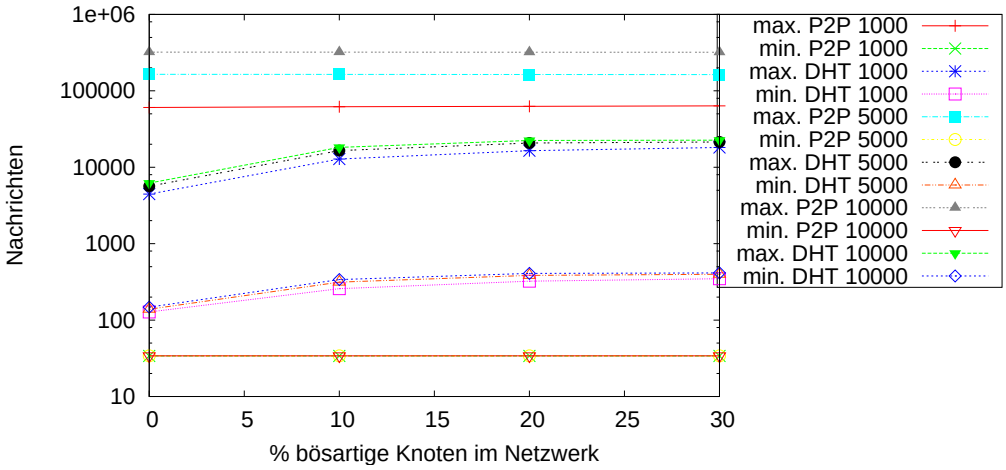
Die Unterschiede zwischen der verlässlichen DHT und dem unstrukturierten P2P-Netzwerk mit aktiver Replikation (siehe Kapitel 5.3) sind für die Unterschiede im Aufwand des Privilegienspeichers verantwortlich. In Abbildung 5.35 werden verschiedene Aktionen zwischen den zwei Varianten des Privilegienspeichers verglichen. In Abbildung 5.35(a) ist zu sehen, dass der Aufwand des DHT-Privilegienspeichers nicht immer geringer ist als der des unstrukturierten P2P-Privilegienspeichers. So ist die dort gezeigte erneute Verschlüsselung von Datenobjekten aufgrund des aufwändigeren Vorgehens beim DHT-Speicher teurer. Während das Maximum beim unstrukturierten P2P-Speicher bei ca. 225 (Minimum 32) Nachrichten liegt, ist dies beim DHT-Speicher bei ca. 600 Nachrichten (Minimum 150) der Fall.

Beim Verändern eines Datenobjekts (Abbildung 5.35(b)) zeigt sich die höhere Effizienz und bessere Skalierbarkeit des DHT-Privilegienspeichers bereits. Beim minimalen Aufwand ist zwar der unstrukturierte P2P-Speicher mit ca. 34 Nachrichten noch besser als der DHT-Speicher mit ca. 150 bis 420 Nachrichten. Bei der maximalen Anzahl der Nachrichten hingegen hat der unstrukturierte P2P-Speicher mit ca. 320000 Nachrichten gegenüber den 6000 bis 25000 Nachrichten des DHT-Speichers klar den höheren Aufwand.

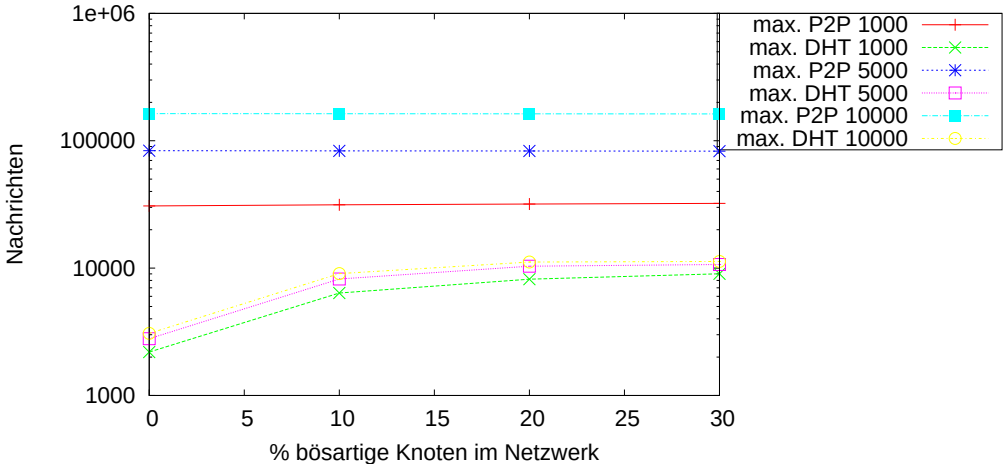
Die häufigsten Anfragen an den Privilegienspeicher betreffen die Durchsetzung von Datenanfragen. Diese sind in Abbildung 5.35(c) dargestellt. Während sie die gleichen minimalen Kosten in Höhe von einer Nachricht besitzen (nicht dargestellt), unterscheiden sie sich deutlich in ihrem maximalen Aufwand. Während der unstrukturierte P2P-Speicher bis zu ca. 163000 Nachrichten bei 10000 Knoten erzeugt, begnügt sich der DHT-Speicher mit ca. 11000 Nachrichten bei 30% bösartiger Peers und nur ca. 3000 Nachrichten bei 0% bösartiger Knoten. Dies zeigt deutlich den wesentlich geringeren Aufwand des DHT-Privilegienspeichers.



(a) Erneute Verschlüsselung von Datenobjekten



(b) Modifikation eines Datenobjekts



(c) Datenanfrage

Abbildung 5.35: Aufwand der clientseitigen Durchsetzung

Wie bereits beim Vergleich der beiden Speichertechniken in Kapitel 5.3 und bei der serverseitigen Durchsetzung in Kapitel 5.5.1 festgestellt, ist die DHT vor allem bei *get* Anfragen wesentlich effizienter als der unstrukturierte P2P-Speicher. Da die Organisation der Privilegien bei beiden Arten des Privilegienspeichers bei der clientseitigen Durchsetzung nahezu identisch ist, kann der unstrukturierte P2P-Speicher auch keine Vorteile aus einer günstigeren Organisation der Privilegien ziehen.

Analog zur serverseitigen Durchsetzung, nimmt der Abstand zwischen dem DHT-Privilegienspeicher und dem unstrukturierten P2P-Privilegienspeicher mit zunehmender Netzwerkgröße aufgrund der besseren Skalierbarkeit der DHT zu. Daneben hat die DHT die Fähigkeit, ihren Aufwand durch den Fehlertest an die im Netzwerk vorhandene Bedrohungslage durch bösartig kooperierende Peers anzupassen. Sie distanziert dadurch den unstrukturierten P2P-Speicher besonders deutlich, wenn keine bösartigen Peers im Netzwerk vorhanden sind.

Bei der clientseitigen Durchsetzung ist zu beachten dass bei der Verwendung des DHT-Privilegienspeichers der Verwaltungsaufwand des Netzwerks höher ist als beim unstrukturierten P2P-Privilegienspeicher, denn der DHT-Privilegienspeicher muss neben der DHT auch noch die DGIO-Gruppe verwalten. Der Aufwand der Netzwerkverwaltung verdoppelt sich dadurch beim DHT-Privilegienspeicher gegenüber dem unstrukturierten P2P-Privilegienspeicher. Insbesondere bei häufigen Netzwerkänderungen ist der DHT-Privilegienspeicher deshalb gegenüber dem unstrukturierten P2P-Privilegienspeicher im Nachteil.

Trotzdem erhält der DHT-Privilegienspeicher auch bei der clientseitigen Durchsetzung durch seine bessere Skalierbarkeit und Effizienz bei der Durchsetzung von Anfragen, klar den Vorzug. Lediglich für kleine, sehr dynamische Netzwerk sollte auf den unstrukturierten P2P-Privilegienspeicher ausgewichen werden.

Selbst bei Verwendung der DHT ist der Aufwand der clientseitigen Durchsetzung erheblich, wie aus den gezeigten Messergebnissen ersichtlich. So verzeichnet eine Datenanfrage in einem 10000 Knoten Netzwerk bis zu 11000 Nachrichten. Schreibende Zugriffe auf Datenobjekte verursachen sogar im schlechtesten Fall 25000 Nachrichten. Diese Werte treten allerdings nur im maximalen Bedrohungsszenario auf.

Hier gilt es aber zu beachten, dass durch die lokale Zwischenspeicherung der Schlüssel dieser Aufwand nur für den ersten Datenzugriff auf ein Objekt anfällt. Das gleiche gilt für die Ermittlung der DGIOs bzw. Stellvertretergruppen. Auch diese können lokal zwischengespeichert werden. Ihre Aktualität kann vor dem Verwenden dann durch direkte Kommunikation mit einer Stichprobe der Stellvertreter überprüft werden, was den Aufwand der Aktionen des Privilegienspeichers ebenfalls beträchtlich reduziert. Zudem ist der in den Graphiken dargestellte maximale Aufwand ohnehin nur nötig, falls sich der Dateneigentümer nicht im Netzwerk befindet. Ist dieser im Netzwerk verfügbar, fällt nur der hier präsentierte minimale Aufwand an, auch wenn der Schlüssel noch nicht im lokalen Zwischenspeicher vorliegt. Bei den für PACS betrachteten Anwendungsbereichen ist davon auszugehen, dass die Mehrzahl der Knoten im Netzwerk verfügbar ist und dadurch die Arbeitslast im Netzwerk erheblich geringer ausfällt.

Aufgrund dieser Optimierungen der für PACS entwickelten clientseitigen Durchsetzung durch Miteinbeziehung der Rahmenbedingungen von PACS ist der Einsatz der clientseitigen Durchsetzung deshalb trotz der enormen maximalen Kosten ohne weiteres möglich.

5.6 Schlussfolgerungen

Die in diesem Kapitel dargestellten Ergebnisse zeigen eindrücklich, dass die DHT den Sicherheitsanforderungen zur Speicherung der Privilegien in keiner Weise genügt. Erst durch die in Chord integrierten Erweiterungen, die in der verlässlichen DHT-Anwendung weitergeführt werden, ist eine zuverlässige Speicherung der Privilegien möglich. Selbiges gilt für das unstrukturierte P2P-Netzwerk, das erst durch die aktive Replikation mit definierter Replikationsgruppe zu einem verlässlichen Privilegienspeicher wird.

Die Komplexitätsanalyse beweist zudem, dass die für PACS entwickelte clientseitige Durchsetzung einen gelungenen Kompromiss zwischen clientseitiger und gruppenbasierter Durchsetzung darstellt. Sie erzeugt wesentlich weniger Aufwand, als die rein gruppenbasierte Durchsetzung. Die Funktionalität ist hierbei lediglich bei der Autorisierungen von Benutzern aufgrund von Kontextinformationen eingeschränkt. Trotz dieser marginalen Einschränkung ist die Funktionalität gegenüber den bisher bekannten clientseitigen Durchsetzungen erheblich erhöht, da die Durchsetzung in PACS erweiterte Zugriffskontrollmodelle unterstützt.

Der abschließende Vergleich der Durchsetzungsvarianten zeigt, dass insbesondere die clientseitiger Durchsetzung einen teilweise grenzwertig hohen Aufwand erzeugt. In Anbetracht der gewonnenen Sicherheit und Funktionalität ist dieser jedoch vertretbar. Der DHT-Privilegienspeicher stellt sowohl bei der clientseitigen als auch der serverseitigen Durchsetzung von PACS die effizientere Alternative für beide Durchsetzungsarten dar.

Zusammenfassend lässt sich anhand der hier präsentierten Ergebnisse die Funktionstüchtigkeit von PACS belegen. Der Aufwand den PACS hierbei erzeugt, schränkt jedoch seinen Anwendungsbereich ein.

Kapitel 6

Zusammenfassung und Ausblick

Die vorliegende Arbeit betrachtet verschiedene Aspekte der Zugriffskontrolle in P2P-Systemen im allgemeinen und P2P-Datenbanken im speziellen. Als Resultat etabliert sie eine Zugriffskontrollkomponente für P2P-Datenbanken (PACS).

6.1 Zusammenfassung

Das Aufkommen von P2P-Netzwerken stellt neue Anforderungen an die Mechanismen, welche die Sicherheit dieser Netzwerke bzw. ihrer Nutzer und deren Daten garantieren. Erste entwickelte Zugriffskontrollsysteme für allgemeine P2P-Systeme bieten noch nicht annähernd die gleiche Funktionalität wie die etablierten Zugriffskontrollsysteme verteilter (und insbesondere föderierter) Datenbanken. Für P2P-Datenbanken selbst sind bislang keine Zugriffskontrollsysteme bekannt. Ziel dieser Arbeit ist es, diesen Zustand zu ändern und damit eine Zugriffskontrolle für P2P-Datenbanken anzubieten, die eine gleichwertige Funktionalität bietet wie jene für föderierte Datenbanken.

Hauptproblem bei der Etablierung einer solchen Zugriffskontrollkomponente für P2P-Umgebungen ist die fehlende zentrale Autorität. Da alle Zugriffskontrollmechanismen für föderierte Datenbanken auf einem zentralen Koordinator als Autorität basieren, lassen sie sich nicht für P2P-Datenbanken einsetzen.

In P2P-Umgebungen müssen die Aufgaben des zentralen Koordinators, d.h. Koordination der lokalen Zugriffskontrollregeln, Verwaltung der globalen Privilegien und Durchsetzung der globalen Privilegien, auf alle Teilnehmer übertragen werden.

Aufgrund der Heterogenität der lokalen Zugriffskontrollkomponenten der Teilnehmer erfolgt für deren Koordination als erster Schritt der Austausch der lokal vorhandenen Zugriffskontrollregeln in einem einheitlichen Format.

Nachdem diese Zugriffskontrollregeln in einem gemeinsamen Austauschformat vorliegen, sollen sie integriert werden. Da in P2P-Umgebungen ein globaler Koordinator fehlt, muss diese Koordination von den einzelnen Mitgliedern vorgenommen werden. Bei der von PACS gewählten Lösung geschieht dies durch Erteilung globaler Privilegien zwischen den Teilnehmern. Die globalen Privilegien werden hierbei gemäß dem globalen Zugriffskontrollmodell erteilt. Die Verwaltung der globalen Privilegien erfordert einen gemeinsamen verteilten Privilegienspeicher. Durch diesen wird die Unterstützung erweiterter, ausdrucksstarker Zugriffskontrollmodelle möglich. Erst mit diesen kann die eingangs angestrebte, im Vergleich zu verteilter Datenbanken gleichwertige Funktionalität erreicht werden.

Neben der verteilten Verwaltung der Privilegien muss auch deren Durchsetzung verteilt erfolgen. Kann auf eine Replikation der Daten verzichtet werden, geschieht dies durch serverseitige Durchsetzung. Ist der Verzicht unmöglich, erfolgt die Durchsetzung clientseitig. Für erweiterte Zugriffskontrollmodelle stellt sich hierbei die Kombination aus clientseitiger Durchsetzung und gruppenbasierter Schlüsselverwaltung als praxistaugliche Lösung heraus.

Eine Übernahme aller Aufgaben des zentralen Koordinators durch eine dezentrale Alternative ist somit möglich. Mit PACS wird für P2P-Datenbanken dadurch zum ersten Mal ein Zugriffskontrollsystem etabliert, das ohne zentrale Komponente auskommt, aber in seiner Funktionalität jener föderierten Datenbanken gleichzusetzen ist.

6.2 Eigene Beiträge und Ergebnisse

Mit PACS wird erstmalig eine dezentral koordinierte Zugriffskontrollkomponente für P2P-Datenbanken entwickelt, die erweiterte Zugriffskontrollmodelle wie RBAC und DAC mit administrativer Delegation unterstützt, ohne auf eine zentrale Komponente zurückzugreifen.

Bislang existierende dezentrale Zugriffskontrollsysteme für P2P-Systeme unterstützen administrative Delegation und RBAC nur ungenügend. Da sie zudem nicht speziell für P2P-Datenbanken entwickelt wurden, nutzen sie vorhandene lokale Zugriffskontrollregeln der Teilnehmer nicht für ihre globale Zugriffskontrolle.

Durch diese Beschränkungen ist das Einsatzgebiet bestehender P2P-Zugriffskontrollsysteme begrenzt. Administrative Delegation und RBAC sind die entscheidenden Techniken, um den Verwaltungsaufwand für die Zugriffsregeln bei großen Nutzerzahlen einzudämmen. Ohne diese Techniken ist eine effiziente Verwaltung der Zugriffsregeln nur mit einer kleinen Nutzerzahl und grobgranularen Zugriffsregeln möglich. Für P2P-Datenbanken, aber auch für die Kapitel 1.2 erwähnten kontrollierten P2P-Netzwerke, sind diese bestehenden Zugriffskontrollsysteme deshalb ungeeignet.

Um diesen Mangel zu beseitigen, wurde PACS entwickelt. Die eigenen Beiträge gestalten sich dabei wie folgt:

Der Ansatz, bestehende Zugriffskontrollregeln für eine globale Zugriffskontrolle zu nutzen, wird in PACS erstmalig auf P2P-Systeme übertragen. Dies erfordert ein gemeinsames, universelles Austauschformat, das in XACML gefunden wurde. Die derzeit noch vorhandenen Beschränkungen von XACML bezüglich der administrativen Delegation, wurden durch Annotation der Rule-Elemente mit den für die Durchsetzung der administrativen Delegation nötigen Informationen überwunden. Die Durchsetzung selbst findet hierbei außerhalb von XACML statt.

Ebenfalls neuartig ist die Idee, diese Zugriffskontrollregeln der XACML Policies durch Erteilung globaler Zugriffskontrollregeln miteinander zu kombinieren. So wird eine globale Zugriffskontrollkomponente etabliert, die ohne einen mit umfassenden Rechten ausgestatteten globalen Administrator („SuperUser“ oder „root“) auskommt.

Hierfür wurde eine dezentrale Verwaltung dieser globalen Privilegien entwickelt, die zur Unterstützung der erweiterten Zugriffskontrollmodelle, wie administrative Delegation in DAC und RBAC, die Bereitstellung eines verlässlichen, verteilten Privilegienspeichers nötig macht. Besonderheit dieser Verwaltung ist hierbei, dass sie auch dann zuverlässig funktioniert, wenn sich bösartig kooperierende Peers im Netzwerk befinden. Diese Anforderungen konnten hierbei von keinem bestehenden verteilten Speicher erfüllt werden. Deshalb wurden für PACS der DHT-basierte Privilegienspeicher und der unstrukturierte P2P-Privilegienspeicher entwickelt.

Beim DHT-basierten Privilegienspeicher konnte hierbei auf schon vorhandene Grundtechniken zu Absicherung strukturierter P2P-Netze zurückgegriffen werden, die vom Autor auf Chord übertragen wurden. Erst so ist die Funktionstüchtigkeit von Chord in Netzwerken mit bösartigen Peers garantiert. Dabei wurde auch eine Erweiterung des Chord Routings realisiert, das nachweislich zu besseren Ergebnissen führt.

Die Grundtechniken zur Absicherung des strukturierten Netzwerks wurden in PACS zudem konsequent auf die DHT-Anwendung und damit den Privilegienspeicher übertragen. Dadurch entsteht erstmals eine verlässliche DHT-Anwendung, die auch bei bis zu 30% bösartig kooperierenden Peers im Netzwerk funktionstüchtig bleibt.

Beim unstrukturierten P2P-Privilegienspeicher wurde eigens für PACS die aktive Replikation mit einer definierten Replikationsgruppe für das unstrukturierte P2P-Netzwerk eingeführt. Die hierfür nötige autonome und trotz bösartigen Knoten und Fluktuation im Netzwerk zuverlässig arbeitende Gruppenverwaltung ist ebenfalls ein eigener Beitrag. Erst sie ermöglicht die verlässliche Speicherung von Daten in unstrukturierten P2P-Netzwerken, trotz des Vorhandenseins von bösartigen Peers.

Um die Integrität der im Privilegienspeicher abgelegten Daten zu schützen, wurden geeignete Gegenmaßnahmen (digitale Signaturen, Versionszähler und selbstverifizierende Objektbezeichner) eingeführt, mit denen Manipulationen erkannt werden können.

Um den Aufwand der dezentralen Durchsetzung des globalen Zugriffskontrollmodells mit administrativer Delegation und RBAC zu optimieren, ist eine optimale Verteilung der Privilegien im Privilegienspeicher nötig. Diese optimale Organisationsform ist abhängig von der Durchsetzungsvariante. PACS unterstützt hierbei neben der serverseitigen auch die clientseitige Durchsetzung.

Durch die in PACS realisierte clientseitige Durchsetzung ist es erstmals möglich, sichere Datenreplikation für erweiterte Zugriffskontrollmodelle anzubieten, die administrative Delegation und RBAC unterstützen. Dies wurde durch die geschickte Kombination von clientseitiger Durchsetzung und einer gruppenbasierten Schlüsselverwaltung erreicht. Die hierfür notwendige autonome Verwaltung der Gruppe ist ein weiterer Beitrag dieser Arbeit. Diese Art der Verwaltung findet auch bei der aktiven Replikation in unstrukturierten Netzwerken Anwendung.

Das Ziel dieser Arbeit war es zu beweisen, dass die Etablierung einer Zugriffskontrolle mit gleichwertiger Funktionalität wie jene föderierter Datenbanken ohne zentralen Koordinator für P2P-Datenbanken möglich ist.

Es gilt nun abschließend zu klären, in wie weit PACS dieser Anforderung gerecht wird. Dies geschieht auf Basis der in Kapitel 1.2 formulierten Zielfragen.

- Kann die zentrale Autorität der globalen Zugriffskontrolle durch dezentrale Mechanismen ersetzt werden?

  Die geschickte Kombination dezentraler Mechanismen zur Verwaltung, Speicherung und Durchsetzung der globalen Privilegien sowie der Authentifizierung von Benutzern ermöglicht es PACS, auf eine zentrale Autorität zu verzichten. Voraussetzung hierfür ist der „Bottom-Up“-Aufbau der Zugriffskontrolle. So sind für die Verwaltung der Privilegien und ihre Durchsetzung, die einzelnen Peers selbst verantwortlich. Für die Speicherung der Privilegien, kann auf vorhandene P2P-Netzwerkspeichertechniken zurückgegriffen werden. Einzige Ausnahme bildet der Teilbereich der Registrierung der Peers im Netzwerk mittels einer PKI. Hier ist zur Abwehr von Sybil-Angriffen eine zentrale Zertifizierungsstelle alternativlos.

- Können bestehende lokal vorhandenen Zugriffskontrollregeln der einzelnen Benutzer zur Reduzierung des Einrichtungsaufwandes für eine globale Zugriffskontrolle genutzt werden?

  PACS baut durch die Verknüpfung der Informationen der Export Policies der Teilnehmer die globale Zugriffskontrolle auf. Durch dieses Vorgehen wird die Nutzung schon bestehender Zugriffskontrollregeln der jeweiligen lokalen Zugriffskontrollkomponenten ein inhärenter Bestandteil von PACS. PACS belegt damit, dass durch den bereits beschriebenen „Bottom-Up"-Aufbau der globalen Zugriffskontrolle und durch ein einheitliches Austauschformat, bestehende lokal vorhandene Zugriffskontrollregeln der einzelnen Benutzer integriert werden können. Als einheitliches Austauschformat zur Überwindung der Heterogenität der lokalen Zugriffskontrollmechanismen, wird hierzu XACML mit Annotationen zur administrativen Delegation festgelegt. Die Übersetzung der lokalen Zugriffskontrollregeln in XACML erfolgt mittels Konverter. Der Bau solcher Konverter ist durch die Ausdruckstärke und Offenheit von XACML prinzipiell möglich. Deren Aufwand und damit Effizienz zu evaluieren, steht noch aus.

- Ist der Einsatz erweiterter Zugriffskontrollmodelle analog zu zentralistischen Lösungen möglich?

  In PACS wurde der Einsatz der erweiterten Zugriffskontrollmodelle am Beispiel der rollenbasierten und der administrativen Delegation bewiesen. Grundvoraussetzung hierfür ist die Implementierung eines gemeinsamen globalen Privilegienspeichers. Die Verwaltung größerer Benutzermengen ist somit effizient möglich.

- Wie müssen diese Privilegien verwaltet, gespeichert und durchgesetzt werden?

  Die Verwaltung der Privilegien ist dezentral organisiert, d.h. sie obliegt den einzelnen Peers. Diese stellen durch die Überprüfung der Autorisierung der Benutzer sicher, dass nur genehmigte Änderungen an den Privilegien vorgenommen werden. Bei der Verwaltung wird darauf geachtet die Privilegien an den Peers abzulegen, die diese später für die Durchsetzung der Privilegien auch benötigen.

  Der geforderte gemeinsame globale Privilegienspeicher wird in PACS durch den unstrukturierten P2P-Privilegienspeicher oder den DHT-Privilegienspeicher realisiert. Der globale Privilegienspeicher stellt die Verfügbarkeit der Privilegien sicher, auch wenn der Daten- bzw. Rolleneigentümer sich nicht im Netzwerk befindet. Dies ist insbesondere für Benutzerzuordnungen und die administrativen Privilegien von Rollen erforderlich. Die abgelegten Privilegien sind vom Grantor signiert und besitzen einen Versionszähler sowie ein Gelöscht-Flag, um sie vor unautorisierten Veränderungen zu schützen und Replay-Angriffe zu erkennen.

  Bei der Durchsetzung werden von PACS die serverseitige Durchsetzung und die clientseitige Durchsetzung eingesetzt. Die serverseitige Durchsetzung arbeitet dabei effizienter, unterstützt aber keine Replikation vertraulicher Daten.

- In wie weit ist ein solcher Zugriffskontrollmechanismus ohne zentrale Autorität gegenüber möglicher im Netzwerk befindlicher bösartiger Knoten fehlertolerant? Wo liegen die Grenzen?

  PACS bietet zwei Angriffspunkte für bösartige Peers, die Verwaltung und die Durchsetzung der Privilegien. Bei der Verwaltung der Privilegien wird sowohl durch den un-

strukturierten P2P-Privilegienspeicher, als auch durch den DHT-Privilegienspeicher die Datensicherheit der darin gespeicherten Privilegien garantiert. Bei der Durchsetzung ist durch den Einsatz schwellenkryptographischer Verfahren sichergestellt, dass ein bösartiger Peer alleine die Vertraulichkeit der Daten nicht verletzten kann. Sowohl das schwellenkryptographischen Verfahren als auch die Privilegienspeicher haben hinsichtlich der Toleranz bösartiger Peers Grenzen. Beim Privilegienspeicher liegt diese bei 30% bösartiger Peers im Netzwerk. Bei den schwellenkryptographischen Verfahren muss die Mehrzahl der Teilnehmer gutartig sein. Aber auch dort empfiehlt sich ein zusätzlicher Sicherheitspuffer von 20%.

- Kann die Verfügbarkeit der Daten durch Datenreplikation garantiert werden, ohne deren Vertraulichkeit zu verletzen?

  Um diese Frage positiv beantworten zu können, wurde für PACS die clientseitige Durchsetzung realisiert. Dabei werden die Daten verschlüsselt im Netzwerk gespeichert. Deren Replikation auf andere Knoten ist damit kein Sicherheitsproblem. Das von PACS eingesetzte Zugriffskontrollmodell wird ebenfalls vollständig unterstützt. Allerdings gibt es dort Einschränkungen bei der Autorisierung von Benutzern aufgrund von Kontextinformationen (sieht Kapitel 3.8). Die durchgeführten Analysen zeigen jedoch, dass die clientseitige Durchsetzung einen sehr hohen Aufwand erzeugt. Grund hierfür ist, dass bei der Verwaltung der Privilegien die Autorisierung einer Änderung bei Abwesenheit des Dateneigentümers durch alle Stellvertreter individuell durchgeführt werden muss.

Datensicherheit und Datenschutz gewinnen im Informationszeitalter, welches durch eine zunehmende Verlagerung der Kommunikation, des Geld- und Warenaustausch in die IT-Welt gekennzeichnet ist, zunehmend an Bedeutung und Brisanz. P2P-Systeme werden vermehrt für den schnellen und unkomplizierten Austausch von Daten genutzt. Die Datensicherheit auch in diesen Umgebung zu gewährleisten, erfordert eine Zugriffskontrolle. Sobald eine Zugriffskontrolle für das Gesamtsystem etabliert werden soll ist eine dezentrale Koordination der Teilnehmer nötig, um die positiven Eigenschaften von P2P-Netzwerken erhalten zu können. Wie eine solche Koordination realisiert werden kann, wurde in PACS gezeigt. Die vorliegende Arbeit belegt klar, dass nicht nur die Speicherung der Daten, sondern auch deren Sicherheit dezentral organisiert werden kann, ohne dass dabei auf Funktionalität und Benutzerkomfort verzichtet werden muss. Datenspeicherung und Datensicherheit müssen nicht länger auf Kosten der Souveränität des Individuums über seine Daten zentralisiert realisiert werden.

Der Verzicht auf einen zentralen Koordinator macht die Verwaltung der Privilegien und deren Durchsetzung aufwändiger. Wann immer auf einen zentralen Koordinator zurückgegriffen werden kann und es keine Sicherheits- und Souveränitätsbedenken gibt, sollte deshalb auf die verteilte Lösung verzichtet werden. In Umgebungen, die trotz dieses zusätzlichen Aufwands nicht auf einen zentralen Koordinator setzen können oder wollen, ist der Einsatz von PACS jedoch ein Fortschritt. Während P2P-Systeme durch ihre Verteilung eine höhere Skalierbarkeit als zentralistische Systeme aufweisen, ist dies bei PACS nicht der Fall. Gründe hierfür sind die geringen Datenmengen, die bei der Privilegienverwaltung anfallen und dass die Überprüfung der Autorisierung keine rechenintensive Datenoperation darstellt. Aus der Verteilung dieser Aufgabe resultieren deshalb keine Performancevorteile. Systeme, die ausschließlich aus Gründen der Skalierbarkeit auf eine P2P-Architektur setzen, profitieren deshalb nur in Ausnahmefällen von PACS.

Darüber hinaus finden Teilbereiche der PACS-Entwicklung auch in anderen Einsatzgebieten Verwendung.

Die clientseitige Durchsetzung, wie sie in PACS realisiert wurde, bietet Nutzern die Möglichkeit, durch die Verschlüsselung ihrer Daten deren Vertraulichkeit eigenverantwortlich sicherzustellen, auch oder gerade wenn Daten zentral auf den Servern eines Dienstanbieter gespeichert werden müssen. Dies ist für Web 2.0 Anwendungen [O'R05] der Fall. Die Verwaltung der Zugriffskontrollregeln in PACS ermöglicht dort selbst bei größeren Benutzermengen eine effiziente Verwaltung dieser Privilegien.

Die verlässliche Datenspeicherung, die PACS in den Privilegienspeichern realisiert, kann in alle P2P-Umgebungen übertragen werden. Dabei kann die Verfügbarkeit der Daten durch die entwickelten Speicher unabhängig von den Daten garantiert werden. Zum Schutz der Datenintegrität müssen an den Daten entsprechend ihrer Semantik Anpassungen vorgenommen werden.

6.3 Ausblick

Die in dieser Arbeit erstellte dezentrale Zugriffskontrolle wurde für P2P-Datenbanken und kontrollierte P2P-Netzwerke entwickelt. Ihr Einsatz ist darüber hinaus auch für andere Anwendungsbereiche denkbar. Dabei sind verteilte Versionsverwaltungssysteme wie git [Git] oder Mercurial [Mer] zu nennen. Auch ein Einsatz in P2P-Anwendungen zum Austausch persönlicher Daten ist denkbar, wie z.B. HomeView [GBGL07] oder vernetzte Personal Information Management Anwendungen [ANR07]. Zuletzt sind als Beispiel noch P2P-basierte, kooperative Webanwendungen zu nennen, wie z.B. Safebook (Cutillo u.a. [CMS09]). Für diese Anwendungen ist entweder noch kein Zugriffskontrollsystem vorhanden oder sie verwenden ein einfaches DAC-Zugriffskontrollmodell ohne administrative Delegation. Die Verwendung von PACS kann in diesem Bereich große Verbesserungen bringen. Hierzu ist zunächst die Erweiterung der prototypischen Implementierung von PACS erforderlich, sodass diese auf einem realen, physischen P2P-Netzwerk basiert. Durch die verwendete Architektur der Simulation ist dies realisierbar.

Angebahnt werden sollte zudem die Verringerung der Einstiegshürde beim Beitritt zum Netzwerk und eine leichtere Wartbarkeit des Systems. Dies kann durch die Entwicklung von Konvertern bzw. Extraktionswerkzeugen für die bekanntesten Datenquellen verwirklicht werden.

Die bisherigen Beschränkungen des XACML-Standards bei administrativer Delegation verhindern dessen Einsatz auf globaler Ebene. Eine Erweiterung des XACML-Standards, und damit eine native Integration administrativer Delegation inklusive Anpassung des Evaluationsprozesses, würde diese überwinden. In diesem Fall könnte auch das globale Zugriffskontrollmodell auf XACML zurückgreifen. Damit wäre die Einheitlichkeit des PACS-Zugriffskontrollmodells gegeben.

Die verlässliche DHT von PACS verwendet Chord als strukturiertes Netzwerk. Chord erfüllt aufgrund seines Aufbaus bereits viele der gestellten Anforderungen in Bezug auf die verlässliche Verwaltung. Ein weiterer Vergleich mit anderen erweiterten strukturierten Netzwerken bezüglich Leistungsfähigkeit und Aufwand wäre hier aufschlussreich. Eventuell ließe sich durch die Verwendung solcher alternativ strukturierter Netzwerke der Aufwand für die verlässliche DHT weiter reduzieren.

Die Organisation und der organisatorische Aufwand bei der Privilegienspeicherung in un-

strukturierten P2P-Netzwerken lässt sich optimieren. Wie die Messergebnisse aufzeigen, ist die Flooding Technik hier noch nicht effizient genug. Es gilt abzuklären, ob redundante Random Walks Weiterleitungen von Nachrichten hier für eine merkliche Reduzierung des Nachrichtenaufkommens sorgen könnten. Allerdings bedarf es zuvor genauer Untersuchungen, wie die Manipulation der Weiterleitung der Nachrichten im Netzwerk durch bösartig kooperierende Peers verhindert werden kann bzw. wie dieser zu begegnen ist.

Die Sicherstellung der Konsistenz in einer verteilten DHT, z.B. durch Unterstützung von Transaktionen, ist eine weitere interessante Forschungsrichtung. Hierdurch können Konsistenz- bzw. Synchronisationsprobleme der Privilegienspeicher genauer betrachtet bzw. gelöst werden.

Bei den Vergleichen verschiedener Durchsetzungsvarianten wurden die unterschiedlichen Operationen wie z.B. Überprüfung der Autorisierung einer Anfrage einander gegenübergestellt. Referenzarbeitslasten für Zugriffskontrollsysteme fehlen bisher. Diese sind notwendig, um die Zugriffskontrollsysteme anhand einer konkreten und repräsentativen Mischung der Operationen zu vergleichen. So liegt ein nächster Arbeitsschritt in der Exploration, Entwicklung und Evaluation realistischer bzw. repräsentativer Arbeitslasten.

Die Frage der Effektivität wird zugunsten der traditionellen, streng hierarchischen Organisationsform mit zentraler Instanz entschieden. In Bezug auf die Sicherheit solcher Systeme ist sie noch unbeantwortet. Eine abschließende Aussage steht noch aus, ob Systeme mit einer zentralen Autorität (und damit einem Angriffspunkt) tatsächlich sicherer sind als Systeme mit einer gleichmäßig, demokratisch verteilten Autorität (und damit vielen kleinen Angriffspunkten). Hier bleibt es interessant, die zukünftige Forschung zu beobachten und voranzutreiben. Diese Arbeit hat aufgezeigt, dass eine vollständig dezentrale Lösung der Zugriffskontrolle auch mit erweiterten Zugriffskontrollmodellen möglich ist. Es liegt am Einsatzgebiet und den jeweiligen Prioritäten der Teilnehmer, ob der erhöhte Aufwand dieser Lösung vertretbar erscheint oder ob Effizienz und Leistungsanforderungen den Ausschlag zugunsten einer zentralen Lösung geben. Persönlich überzeugt mich der Gedanke, durch die erreichte, vollkommen dezentrale Struktur immer die letzte Autorität für die eigenen publizierten Daten zu behalten. Dies steht im Gegensatz zu allen zentralisierten Lösungen, bei denen sich die einzelnen Benutzer letztlich einer übergeordneten Instanz unterordnen müssen.

Liste der Abkürzungen

ACL	Access Control List
AES	Advance Encryption Standard
API	Application Programming Interface
CAN	Content Addressable Network
CPU	Central Processing Unit
DAC	Discretionary Access Control
DBMS	Datenbank Management System
DGIO	Delegation Group Information Object
DHT	Verteilte Hashtabelle
EPFL	Ecole Polytechnique Fédérale de Lausanne
ETHZ	Eidgenössische Technische Hochschule Zürich
MAC	Mandatory Access Control
MD5	Message-Digest Algorithm 5
P2P	Peer-to-Peer
PACS	Peer Access Control System
PDMS	Peer Data Management System
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PKI	Public Key Infrastruktur
RBAC	Rollenbasierte Zugriffskontrolle
SD	Schlüsseldatei
SQL	Structured Query Language
TCT	Trusted Computing Technology
TPM	Trusted Platform Modul
UPDF	Unified Peer-to-Peer Database Framework
UZH	Universität Zürich
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

Literaturverzeichnis

[ACMD+03a] Aberer, Karl, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva und Roman Schmidt: *P-Grid: A Self-organizing Structured P2P System.* SIGMOD Rec., 32(3):29–33, 2003.

[ACMD+03b] Aberer, Karl, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, Roman Schmidt und Jie Wu: *Advanced Peer-to-Peer Networking: The P-Grid System and its Applications.* PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems, 2003.

[ACMHP04] Aberer, Karl, Philippe Cudré-Mauroux, Manfred Hauswirth und Tim Van Pelt: *GridVine: Building Internet-Scale Semantic Overlay Networks.* In: *ISWC '04: Proceedings of the International Semantic Web Conference*, Seiten 107–121, 2004.

[ACTT01] Agarwal, D.A., O. Chevassut, M.R. Thompson und G. Tsudik: *An Integrated Solution for Secure Group Communication in Wide-Area Networks.* In: *Proceedings of the Sixth IEEE Symposium on Computers and Communications*, Seiten 22–28, 2001.

[AD01] Aberer, Karl und Zoran Despotovic: *Managing Trust in a Peer-2-Peer Information System.* In: *CIKM '01: Proceedings of the Tenth International Conference on Information and Knowledge Management*, Seiten 310–317, 2001.

[ADDS07] Ardagna, Claudio A., Ernesto Damiani, Sabrina De Capitani di Vimercati und Pierangela Samarati: *XML Security.* In: Petkovic, Milan und Williem Jonker (Herausgeber): *Security, Privacy and Trust in Modern Data Management*, Seiten 71–86. Springer-Verlag New York, Inc., 2007.

[Ake] *Webseite Akenti.* `http://acs.lbl.gov/software/Akenti/`. Abgerufen am 18.06.2010.

[AKK+03] Arenas, Marcelo, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller und John Mylopoulos: *The Hyperion Project: From Data Integration to Data Coordination.* SIGMOD Rec., 32(3):53–58, 2003.

[ANR07] Albrecht, Alexander, Felix Naumann und Armin Roth: *Networked PIM using PDMS.* In: *NETB'07: Proceedings of the 3rd USENIX international*

workshop on Networking meets databases, Seiten 1–6, Berkeley, CA, USA, 2007. USENIX Association.

[APHS02] Aberer, Karl, Magdalena Punceva, Manfred Hauswirth und Roman Schmidt: *Improving Data Access in P2P Systems.* IEEE Internet Computing, 6(1):58–67, 2002.

[APV07] Akbarinia, Reza, Esther Pacitti und Patrick Valduriez: *Data Currency in Replicated DHTs.* In: *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Seiten 211–222, New York, NY, USA, 2007. ACM.

[ATS04] Androutsellis-Theotokis, Stephanos und Diomidis Spinellis: *A Survey of Peer-to-Peer Content Distribution Technologies.* ACM Comput. Surv., 36(4):335–371, 2004.

[Bac10] Bachfeld, Daniel: *Erneute Datenpanne bei AT&T.* `http://www.heise.de/newsticker/meldung/Erneute-Datenpanne-bei-AT-T-1023437.html`, Juni 2010. Abgerufen am 18.06.2010.

[BAS04] Bharambe, Ashwin R., Mukesh Agrawal und Srinivasan Seshan: *Mercury: Supporting Scalable Multi-Attribute Range Queries.* SIGCOMM Comput. Commun. Rev., 34(4):353–366, 2004.

[BCF01] Bertino, Elisa, Silvana Castano und Elena Ferrari: *Securing XML Documents with Author-X.* IEEE Internet Computing, 5(3):21–31, 2001.

[BCF+04] Bertino, Elisa, Barbara Carminati, Elena Ferrari, Bhavani Thuraisingham und Amar Gupta: *Selective and Authentic Third-Party Distribution of XML Documents.* IEEE Transactions on Knowledge and Data Engineering, 16(10):1263–1278, 2004.

[BCOS08] Bonifati, Angela, Panos K. Chrysanthis, Aris M. Ouksel und Kai-Uwe Sattler: *Distributed Databases and Peer-to-Peer Databases: Past and Present.* SIGMOD Rec., 37(1):5–11, 2008.

[BEM04] Berket, Karlo, Abdelilah Essiari und Artur Muratas: *PKI-Based Security for Peer-to-Peer Information Sharing.* In: *P2P '04: Proceesings of the Fourth International Conference on Peer-to-Peer Computing*, Seiten 45–52, 2004.

[BGK+02] Bernstein, Philip A., Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini und Ilya Zaihrayeu: *Data Management for Peer-to-Peer Computing: A Vision.* In: *WebDB '02: Proceedings of the Fifth International Workshop on the Web and Databases*, Seiten 89–94, June 2002.

[Bis08] Biskup, Joachim: *Security in Computing Systems: Challenges, Approaches and Solutions.* Springer Publishing Company, Incorporated, 2008.

[BK04] Bearly, Terry und Vijay Kumar: *Expanding Trust Beyond Reputation in Peer-To-Peer Systems.* In: *DEXA '04: Proceedings of the 15th International Workshop on Database and Expert Systems Applications*, Seiten 966–970, 2004.

[BL73] Bell, D. Elliott und Leonard J. LaPadula: *Secure Computer Systems: Mathematical Foundations.* Technischer Bericht 2547, MITRE, März 1973.

[BN84] Birrell, Andrew D. und Bruce Jay Nelson: *Implementing Remote Procedure Calls.* ACM Trans. Comput. Syst., 2(1):39–59, 1984.

[BNP04] Bouganim, Luc, François Dang Ngoc und Philippe Pucheral: *Client-based access control management for XML documents.* In: *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, Seiten 84–95. VLDB Endowment, 2004.

[Bra10] Braun, Herbert: *Datenpanne bei Facebook.* `http://www.heise.de/newsticker/meldung/Datenpanne-bei-Facebook-968117.html`, März 2010. Abgerufen am 18.06.2010.

[CAMN03] Cuenca-Acunam, Francisco Matias, Richard P. Martin und Thu D. Nguyen: *Autonomous Replication for High Availability in Unstructured P2P Systems.* In: *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, Seiten 99–108, Oktober 2003.

[CC07] Chen, Yu und Wei Chen: *Decentralized, Connectivity-Preserving, and Cost-Effective Structured Overlay Maintenance.* In: *SSS '07: Proceedings of the 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Seiten 97–113. Springer, 2007.

[CDG+02] Castro, Miguel, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron und Dan S. Wallach: *Secure routing for structured peer-to-peer overlay networks.* SIGOPS Oper. Syst. Rev., 36(SI):299–314, 2002.

[CDHR03] Castro, Miguel, Peter Druschel, Y. Charlie Hu und Antony Rowstron: *Proximity neighbor selection in tree-based structured peer-to-peer overlays.* Technischer Bericht MSR-TR-2003-52, Microsoft Research, 2003.

[CdVF97] Castano, S., S. De Capitani di Vimercati und M.G. Fugini: *Automated Derivation of Global Authorizations for Database Federations.* Journal of Computer Security, 5(4):271–301, 1997.

[Cel] *celeste.* `https://celeste.dev.java.net/`. Abgerufen am 18.06.2010.

[CFMS94] Castano, Silvana, Maria Grazia Fugini, Giancarlo Martella und Pierangela Samarati: *Database Security.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1994.

[Cho] *The Chord/DHash Project.* `http://pdos.csail.mit.edu/chord/`. Abgerufen am 18.06.2010.

[CL06] Chen, Wei und Xuezheng Liu: *Enforcing Routing Consistency in Structured Peer-to-Peer Overlays: Should We and Could We?* In: *IPTPS '06: Proceedings from the Fifth International Workshop on Peer-to-Peer Systems*, 2006.

[CLGS04] Crainiceanu, Adina, Prakash Linga, Johannes Gehrke und Jayavel Shanmugasundaram: *Querying Peer-to-Peer Networks Using P-Trees.* In: *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, Seiten 25–30. ACM Press, 2004.

[CLM+04] Crainiceanu, Adina, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke und Jayavel Shanmugasundaram: *An Indexing Framework for Peer-to-Peer Systems.* In: *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Seiten 939–940. ACM Press, 2004.

[CLM+07] Crainiceanu, Adina, Prakash Linga, Ashwin Machanavajjhala, Johannes Gehrke und Jayavel Shanmugasundaram: *P-Ring: An Efficient and Robust P2P Range Index Structure.* In: *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Seiten 223–234, New York, NY, USA, 2007. ACM.

[CMAA07] Cudré-Mauroux, Philippe, Suchit Agarwal und Karl Aberer: *GridVine: An Infrastructure for Peer Information Management.* IEEE Internet Computing, 11(5):36–44, 2007.

[CMAB+07] Cudré-Mauroux, Philippe, Suchit Agarwal, Adriana Budura, Parisa Haghani und Karl Aberer: *Self-Organizing Schema Mappings in the GridVine Peer Data Management System.* In: *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, Seiten 1334–1337. VLDB Endowment, 2007.

[CMAF06] Cudré-Mauroux, Philippe, Karl Aberer und Andras Feher: *Probabilistic Message Passing in Peer Data Management Systems.* In: *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, Washington, DC, USA, 2006. IEEE Computer Society.

[CMS09] Cutillo, Leucio Antonio, Refik Molva und Thorsten Strufe: *Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-Life Trust.* Communications Magazine, IEEE, 47(12):94 –101, December 2009.

[Com05a] Committee, OASIS Access Control Technical: *Core and Hierachical Role Based Access Control (RBAC) Profile of XACML v2.0.* `http://docs.oasis-open.org/xacml/2.0/access_control- xacml-2.0-rbac-profile1-spec-os.pdf`, 2005. Abgerufen am 18.06.2010.

[Com05b] Committee, OASIS Access Control Technical: *eXtensible Access Control Markup Language (XACML) Version 2.0.* `http://docs.oasis-open.org/xacml/2.0/access_control- xacml-2.0-core-spec-os.pdf`, 2005. Abgerufen am 18.06.2010.

[Con97] Conrad, Stefan: *Föderierte Datenbanksysteme: Konzepte der Datenintegration.* Springer Verlag, Berlin, Heidelberg, New York, 1997.

[CS02] Cohen, Edith und Scott Shenker: *Replication strategies in unstructured peer-to-peer networks.* SIGCOMM Comput. Commun. Rev., 32(4):177–190, 2002.

[CSi] *Mesquite Software – Development Toolkit.* `http://www.mesquite.com/`. Abgerufen am 18.06.2010.

[CSMB05] Crispo, Bruno, Swaminathan Sivasubramanian, Pietro Mazzoleni und Elisa Bertino: *P-Hera: Scalable fine-grained access control for P2P infrastructures.* In: *ICPADS '05: Proceedings of the 11th International Conference on Parallel and Distributed Systems*, Seiten 585–591, 2005.

[DA06] Despotovic, Zoran und Karl Aberer: *P2P reputation management: Probabilistic estimation vs. social networks.* Comput. Netw., 50(4):485–500, 2006.

[DBK+01] Dabek, Frank, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica und Hari Balakrishnan: *Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service.* In: *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, Seiten 81–86, May 2001.

[DF89] Desmedt, Yvo und Yair Frankel: *Threshold cryptosystems.* In: *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 307–315, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[DF92] Desmedt, Yvo und Yair Frankel: *Shared Generation of Authenticators and Signatures.* In: *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 457–469, London, UK, 1992. Springer-Verlag.

[DGBC04] Detsch, A., L. P. Gaspary, M. P. Barcellos und G. G. H. Cavalheiro: *Towards a Flexible Security Framework for Peer-to-Peer based Grid Computing.* In: *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, Seiten 52–56, New York, NY, USA, 2004. ACM.

[DH06] Dinger, Jochen und Hannes Hartenstein: *Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration.* In: *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, Seiten 756–763, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[DHA03] Datta, Anwitaman, Manfred Hauswirth und Karl Aberer: *Updates in Highly Unreliable, Replicated Peer-to-Peer Systems.* In: *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[DHT04] Dong, Xin, Alon Y. Halevy und Igor Tatarinov: *Containment of Nested XML Queries.* In: *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, Seiten 132–143. VLDB Endowment, 2004.

[DKK+01] Dabek, Frank, M. Frans Kaashoek, David Karger, Robert Morris und Ion Stoica: *Wide-area cooperative storage with CFS.* In: *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, Seiten 202–215, New York, NY, USA, 2001. ACM.

[Dou01] Douceur, John R.: *The Sybil Attack.* In: *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Seiten 251–260, 2001.

[DR01] Druschel, P. und A. Rowstron: *PAST: a large-scale, persistent peer-to-peer storage utility.* In: *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, 2001*, Seiten 75–80, Mai 2001.

[DR02] Daemen, Joan und Vincent Rijmen: *The Design of Rijndael.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[dSGBD05] Silva, Juliano F. da, Luciano P. Gaspary, Marinho P. Barcellos und Andre Detsch: *Policy-Based Access Control in Peer-to-Peer Grid Systems.* In: *Grids 2005: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, Seiten 107–113, November 2005.

[dVFS07] Vimercati, Sabrina De Capitani di, Sara Foresti und Pierangela Samarati: *Authorization and Access Control.* In: Petkovic, Milan und Williem Jonker (Herausgeber): *Security, Privacy and Trust in Modern Data Management*, Seiten 39–53. Springer-Verlag New York, Inc., 2007.

[dVS96] Vimercati, Sabrina De Capitani di und Pierangela Samarati: *Access Control in Federated Systems.* In: *NSPW '96: Proceedings of the 1996 workshop on New security paradigms*, Seiten 87–99, New York, NY, USA, 1996. ACM Press.

[dVS97] Vimercati, S. De Capitani di und P. Samarati: *Authorization Specification and Enforcement in Federated Database Systems.* Journal of Computer Security, 5(2):155–188, 1997.

[DZD+03] Dabek, Frank, Ben Zhao, Peter Druschel, John Kubiatowicz und Ion Stoica: *Towards a Common API for Structured Peer-to-Peer Overlays.* In: *IPTPS '03: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Seiten 33–44. Springer Berlin / Heidelberg, 2003.

[EAABH03] El-Ansary, Sameh, Luc Onana Alima, Per Brand und Seif Haridi: *Efficient Broadcast in Structured P2P Networks.* In: *IPTPS '03: Proceedings of the Second International Workshop Peer-to-Peer Systems*, Seiten 304–314, 2003.

[EN07] Elmasri, Ramez und Shamkant B. Navathe: *Fundamentals of Database Systems.* Pearson, 5. Auflage, 2007.

[Fac] *Willkommen bei Facebook.* `http://www.facebook.com/`. Abgerufen am 18.06.2010.

[Fag78] Fagin, Ronald: *On an Authorization Mechanism.* ACM Trans. Database Syst., 3(3):310–319, 1978.

[Far02] Farrell, S.: *An Internet Attribute Certificate Profile for Authorization.* Request for Comments 3281, April 2002.

[FCAG03] Ferraiolo, David F., R. Chandramouli, Gail-Joon Ahn und Serban I. Gavrila: *The Role Control Center: Features and Case Studies.* In: *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, Seiten 12–20, New York, NY, USA, 2003. ACM.

[FK92] Ferraiolo, David und Richard Kuhn: *Role-Based Access Control.* In: *15th NIST-NCSC National Computer Security Conference*, Seiten 554–563, 1992.

[FKC07] Ferraiolo, David F., D. Richard Kuhn und Ramaswamy Chandramouli: *Role-Based Access Control.* Information Security and Privacy. Artech House Publishers, 2. Auflage, 2007.

[FKLZ04a] Franconi, Enrico, Gabriel M. Kuper, Andrei Lopatenko und Ilya Zaihrayeu: *The coDB Robust Peer-to-Peer Database System.* In: *Proceedings of the Twelfth Italian Symposium on Advanced Database Systems, SEBD*, Seiten 382–393, 2004.

[FKLZ04b] Franconi, Enrico, Gabriel M. Kuper, Andrei Lopatenko und Ilya Zaihrayeu: *Queries and Updates in the coDB Peer to Peer Database System.* In: *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, Seiten 1277–1280. VLDB Endowment, 2004.

[Fli] *Willkommen bei Flickr – Fotosharing.* `http://www.flickr.com/`. Abgerufen am 18.06.2010.

[Frea] *FreePastry.* `http://www.freepastry.org/FreePastry/`. Abgerufen am 18.06.2010.

[Freb] *Pastry – A scalable, decentralized, self-organizing and fault-tolerant substrate for peer-to-peer applications.* `http://www.freepastry.org/`. Abgerufen am 18.06.2010.

[Gam85] Gamal, Taher El: *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Transactions on Information Theory, 31(4):469–472, Juli 1985.

[GBGL07] Geambasu, Roxana, Magdalena Balazinska, Steven D. Gribble und Henry M. Levy: *HomeViews: Peer-to-Peer Middleware for Personal Data Sharing Applications.* In: *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, Seiten 235–246, New York, NY, USA, 2007. ACM.

[GD71] Graham, G. Scott und Peter J. Denning: *Protection: principles and practice.* In: *AFIPS '71 (Fall): Proceedings of the November 16-18, 1971, fall joint computer conference*, Seiten 417–429, New York, NY, USA, 1971. ACM.

[Git] *Git – Fast Version Control System.* `http://git-scm.com/`. Abgerufen am 18.06.2010.

[Gon93] Gong, Li: *Increasing Availability and Security of an Authentication Service.* IEEE Journal on Selected Areas in Communications, 11(5):657–662, Jun 1993.

[Gri] *GridVine.* `http://lsirwww.epfl.ch/GridVine/`. Abgerufen am 18.06.2010.

[GRJK00] Gennaro, Rosario, Tal Rabin, Stanislav Jarecki und Hugo Krawczyk: *Robust and Efficient Sharing of RSA Functions.* Journal of Cryptology, 13(2):273–300, 2000.

[Gro03] Grothoff, Christian: *An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks.* Wirtschaftsinformatik, 3-2003, Juni 2003.

[Gro07] Groffen, Fabian: *Armada: A Symbiosis of Traditional Distributed Databases and P2P Systems.* In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, September 2007. PhD Workshop.

[Gro09] Groffen, Fabian: *Armada An Evolving Database System.* Doktorarbeit, CWI, the Dutch national research centre for mathematics and computer science, 2009.

[GSMB03] Goh, Eu-Jin, Hovav Shacham, Nagendra Modadugu und Dan Boneh: *SiRiUS: Securing Remote Untrusted Storage.* In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*, 2003.

[GW76] Griffiths, Patricia P. und Bradford W. Wade: *An Authorization Mechanism for a Relational Database System.* ACM Trans. Database Syst., 1(3):242–255, 1976.

[GY04] Gummadi, Abhilash und Jong P. Yoon: *Modeling Group Trust For Peer-to-Peer Access Control.* In: *DEXA '04: Proceedings of the 15th International Workshop on Database and Expert Systems Applications*, Seiten 971–978, 2004.

[HCH+05] Huebsch, Ryan, Brent N. Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica und Aydan R. Yumerefendi: *The Architecture of PIER: an Internet-Scale Query Processor.* In: *Proceedings of the 2005 CIDR Conference*, Seiten 28–43, 2005.

[Hel03] Hellerstein, Joseph M.: *Toward Network Data Independence.* SIGMOD Rec., 32(3):34–40, 2003.

[HHB+03] Huebsch, Ryan, Joseph M. Hellerstein, Nick Lanham Boon, Thau Loo, Scott Shenker und Ion Stoica: *Querying the Internet with PIER.* In: *VLDB '03: Proceedings of the 29th international conference on Very large data bases*, Seiten 321–332. VLDB Endowment, 2003.

[HHNR05] Heese, Ralf, Sven Herschel, Felix Naumann und Armin Roth: *Self-Extending Peer Data Management.* In: *Proceedings Datanbanksysteme in Business, Technologie und Web (BTW) 2005*, Band 11, Seiten 145–164, 2005.

[HILM02] Hacigümüş, Hakan, Bala Iyer, Chen Li und Sharad Mehrotra: *Executing SQL over Encrypted Data in the Database-Service-Provider Model.* In: *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, Seiten 216–227, New York, NY, USA, 2002. ACM.

[HIM+04] Halevy, Alon Y., Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu und Igor Tatarinov: *The Piazza Peer Data Management System.* IEEE Transactions on Knowledge and Data Engineering, 16(7):787–798, July 2004.

[HIST03] Halevy, Alon Y., Zachary G. Ives, Dan Suciu und Igor Tatarinov: *Schema Mediation in Peer Data Management Systems.* In: *ICDE '03: Proceedings of the 19th International Conference on Data Engineering*, Seiten 505–516, 2003.

[HJKY95] Herzberg, Amir, Stanislaw Jarecki, Hugo Krawczyk und Moti Yung: *Proactive Secret Sharing Or: How to Cope With Perpetual Leakage.* In: *CRYPTO '95: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 339–352, 1995.

[HJPPW01] Hagström, Åsa, Sushil Jajodia, Francesco Parisi-Presicce und Duminda Wijesekera: *Revocations-A Classification.* In: *CSFW '01: Proceedings of the 14th IEEE workshop on Computer Security Foundations*, Seiten 44–58, Washington, DC, USA, 2001. IEEE Computer Society.

[HM85] Heimbigner, Dennis und Dennis McLeod: *A federated architecture for information management.* ACM Trans. Inf. Syst., 3(3):253–278, 1985.

[Hos02] Hoschek, Wolfgang: *A Unified Peer-to-Peer Database Framework for Scalable Service and Resource Discovery.* In: *Proceedings of the Third International Workshop on Grid Computing*, Seiten 126–144. Springer-Verlag, 2002.

[HRU76] Harrison, Michael A., Walter L. Ruzzo und Jeffrey D. Ullman: *Protection in operating systems.* Commun. ACM, 19(8):461–471, 1976.

[HT88] Herlihy, Maurice und J. D. Tygar: *How to Make Replicated Data Secure.* In: *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, Seiten 379–391, London, UK, 1988. Springer-Verlag.

[JA06] Jin, Jing und Gail-Joon Ahn: *Role-based Access Management for Ad-hoc Collaborative Sharing.* In: *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, Seiten 200–209, New York, NY, USA, 2006. ACM Press.

[Jav] *JavaSim – Home.* `http://javasim.codehaus.org/`. Abgerufen am 18.06.2010.

[JD94] Jonscher, Dirk und Klaus R. Dittrich: *An Approach for Building Secure Database Federations.* In: *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, Seiten 24–35, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[JMJV] Jelasity, Márk, Alberto Montresor, Gian Paolo Jesi und Spyros Voulgaris: *The Peersim Simulator.* `http://peersim.sf.net`.

[Jon98] Jonscher, Dirk: *Access Control in Object-Oriented Federated Database Systems.* Doktorarbeit, Universität Zürich, 1998.

[JOT+06] Jagadish, H. V., Beng Chin Ooi, Kian-Lee Tan, Quang Hieu Vu und Rong Zhang: *Speeding up Search in Peer-to-Peer Networks with A Multi-way Tree Structure.* In: *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, Seiten 1–12, New York, NY, USA, 2006. ACM.

[JOV05] Jagadish, H. V., Beng Chin Ooi und Quang Hieu Vu: *BATON: A Balanced Tree Structure for Peer-to-Peer Networks.* In: *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, Seiten 661–672. VLDB Endowment, 2005.

[JP03] Josef Pieprzyk, Thomas Hardjono, Jennifer Seberry: *Fundamentals of Computer Security.* Springer-Verlag, 2003.

[JUn] *JUnit.org.* `http://www.junit.org/`. Abgerufen am 18.06.2010.

[KAM03a] Kementsietsidis, Anastasios, Marcelo Arenas und Renée J Miller: *Managing Data Mappings in the Hyperion Project.* In: *ICDE '03: Proceedings of the 19th International Conference on Data Engineering*, Seiten 732–734, March 2003.

[KAM03b] Kementsietsidis, Anastasios, Marcelo Arenas und Renée J. Miller: *Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues.* In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, Seiten 325–336, New York, NY, USA, 2003. ACM Press.

[KBC07] Koç, Erol, Marcel Baur und Germano Caronni: *PACISSO: P2P Access Control Incorporating Scalability and Self-Organization for Storage Systems.* Technischer Bericht TR-2007-165, Sun Microsystems, Inc., June 2007.

[KKM+03] Kantere, Vasiliki, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis und Marcelo Arenas: *Coordinating Peer Databases Using ECA Rules.* In: *Revised Papers First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, Seiten 108–122, 2003.

[KLL+97] Karger, David, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine und Daniel Lewin: *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web.* In: *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, Seiten 654–663, New York, NY, USA, 1997. ACM.

[Koç06] Koç, Erol: *Access Control in Peer-to-Peer Storage Systems.* Diplomarbeit, ETH Zurich, October 2006.

[KP04] Koloniari, Goergia und Evaggelia Pitoura: *Content-Based Routing of Path queries in Peer-to-Peer Systems.* In: *EDBT '04: Proceedings of the 9th International Conference on Extending Database Technology.* Springer, März 2004.

[Kra93] Krawczyk, Hugo: *Distributed fingerprints and secure information dispersal.* In: *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*, Seiten 207–218, New York, NY, USA, 1993. ACM.

[Kre09] Kremp, Matthias: *Sicherheitslücke – Neues Datenleck bei SchülerVZ aufgedeckt.* `http://www.spiegel.de/netzwelt/web/0,1518,657800,00.html`, Oktober 2009. Abgerufen am 28.11.2009.

[KSGM03a] Kamvar, Sepandar D., Mario T. Schlosser und Hector Garcia-Molina: *The EigenTrust Algorithm for Reputation Management in P2P Networks.* In: *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, Seiten 640–651, New York, NY, USA, 2003. ACM Press.

[KSGM03b] Kamvar, Sepandar D., Mario T. Schlosser und Hector Garcia-Molina: *Incentives for Combatting Freeriding on P2P Networks.* In: *Euro-Par 2003 Parallel Processing: Proceedings of the 9th International Euro-Par Conference*, Seiten 1273–1279, 2003.

[Lam74] Lampson, Butler W.: *Protection.* SIGOPS Oper. Syst. Rev., 8(1):18–24, 1974.

[LCC+02] Lv, Qin, Pei Cao, Edith Cohen, Kai Li und Scott Shenker: *Search and Replication in Unstructured Peer-to-Peer Networks.* In: *ICS '02: Proceedings of the 16th international conference on Supercomputing*, Seiten 84–95, New York, NY, USA, 2002. ACM.

[LM91] Lai, Xuejia und James L. Massey: *A Proposal for a New Block Encryption Standard.* In: *EUROCRYPT '90: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, Seiten 389–404, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[LN06] Leser, Ulf und Felix Naumann: *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.* dpunkt, 2006.

[LNBK02] Liben-Nowell, David, Hari Balakrishnan und David Karger: *Analysis of the Evolution of Peer-to-Peer Systems.* In: *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, Seiten 233–242. ACM Press, 2002.

[LPM+04] López, Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol Ahulló, Helio Tejedor und Robert Rallo: *PlanetSim: A New Overlay Network Simulation Framework.* In: *SEM '04: Revised Selected Papers of the 4th International Workshop on Software Engineering and Middleware*, Seiten 123–136. Springer, September 2004.

[MCMP04] Mukkamala, Ravi, L. Chekuri, Mohammed Moharrum und S. Palley: *Policy-based Security Management for Enterprise Systems.* In: *DBSEC '04: Proceedings of the 18th Annual IFIP WG 11.3 Working Conference Data and Applications Security XVIII*, Seiten 219–233. Kluwer, 2004.

[MCSB05] Mazzoleni, Pietro, Bruno Crispo, Swaminathan Sivasubramanian und Elisa Bertino: *Efficient Integration of Fine-grained Access Control in Large-scale Grid Services.* In: *SCC '05: Proceedings of the IEEE International Conference on Services Computing*, Band 1, Seiten 77–84, 2005.

[Mer] *Mercurial SCM.* `http://mercurial.selenic.com/`. Abgerufen am 18.06.2010.

[MKRL+03] Milojicic, Dejan S., Vana Kalogeraki, Kiran Nagaraja Rajan Lukose, Jim Pruyne, Bruno Richard, Sami Rollins und Zhichen Xu: *Peer-to-Peer Computing.* Technischer Bericht HPL-2002-57 (R.1), HP Laboratories Palo Alto, Juli 2003.

[MS03] Miklau, Gerome und Dan Suciu: *Controlling Access to Published Data Using Cryptography.* In: *VLDB '03: Proceedings of the 29th international conference on Very large data bases*, Seiten 898–909. VLDB Endowment, 2003.

[MyS] *MySpace.* `http://www.myspace.com/`. Abgerufen am 18.06.2010.

[Nat99] National Institute of Standards and Technology: *Data Encryption Standard (DES).* Federal Information Processing Standards Publication (FIPS PUB) 46-3, U.S. Department of Commerce, Oktober 1999.

[Nat08] National Institue of Standards and Technology: *Secure Hash Standard (SHS).* Federal Information Processing Standards Publication (FIPS PUB) 180-3, U.S. Department of Commerce, Oktober 2008.

[Nat09] National Institute of Standards and Technology: *Digital Signature Standard (DSS).* Federal Information Processing Standards Publication (FIPS PUB) 186-3, U.S. Department of Commerce, Juni 2009.

[NBL+06] Naicken, Stephen, Anirban Basu, Barnaby Livingston, Sethalat Rodhetbhai und Ian Wakeman: *Towards Yet Another Peer-to-Peer Simulator.* In: *Proceedings of The Fourth International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs), Ilkley, UK*, 2006.

[NBL+07] Naicken, Stephen, Anirban Basu, Barnaby Livingston, Sethalat Rodhetbhai, Ian Wakeman und Dan Chalmers: *The State of Peer-to-Peer Simulators and Simulations.* SIGCOMM Comput. Commun. Rev., 37(2):95–98, 2007.

[NBLR06] Naicken, Stephen, Anirban Basu, Barnaby Livingston und Sethalat Rodhetbhai: *A Survey of Peer-to-Peer Network Simulators.* In: *PGNet '06: Proceedings of the 7th Annual Postgraduate Symposium*, Liverpool, UK, 2006. EPSRC.

[NIS04] NIST: *Role Based Access Control.* ANSI INCITS 359-2004, Februar 2004.

[NOTZ03] Ng, Wee Siong, Beng Chin Ooi, Kian-Lee Tan und Aoying Zhou: *PeerDB: A P2P-based System for Distributed Data Sharing.* In: *ICDE '03: Proceedings of the 19th International Conference on Data Engineering*, Seiten 633–644, 2003.

[Nsn] *Nsnam.* `http://nsnam.isi.edu/nsnam/index.php/Main_Page`. Abgerufen am 18.06.2010.

[NT04] Ntarmos, Nikos und Peter Triantafillou: *SeAl: Managing Acesses and Data in Peer-to-Peer Data Sharing Networks.* In: *P2P '04: Proceeding of the Fourth International Conference on Peer-to-Peer Computing*, Seiten 116–123, 2004.

[NTY03] Narasimha, Maithili, Gene Tsudik und Jeong Hyun Yi: *On the Utility of Distributed Cryptography in P2P and MANETs: The Case of Membership Control.* In: *Proceedings of the 11th IEEE International Conference on Network Protocols 2003*, Seiten 336–345. IEEE Computer Society, 2003.

[NW98] Naor, Moni und Avishai Wool: *Access Control and Signatures via Quorum Secret Sharing.* IEEE Trans. Parallel Distrib. Syst., 9(9):909–922, 1998.

[O'R05] O'Reilly, Tim: *What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software.* `http://www.oreillynet.com/pub/a/oreilly/tim/ news/2005/09/30/what-is-web-20.html`, 2005. Abgerufen am 18.06.2010.

[ORMN05] Olmedilla, Daniel, Omer F. Rana, Brian Matthews und Wolfgang Nejdl: *Security and Trust Issues in Semantic Grids.* In: *Semantic Grid: The Convergence of Technologies*, Dagstuhl Seminar Proceedings, 2005.

[OS02] Oh, Sejong und Ravi Sandhu: *A Model for Role Administration Using Organization Structure.* In: *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, Seiten 155–162, New York, NY, USA, 2002. ACM.

[Osb07] Osborn, Sylvia L.: *Role-Based Access Control.* In: Petkovic, Milan und Williem Jonker (Herausgeber): *Security, Privacy and Trust in Modern Data Management*, Seiten 55–70. Springer-Verlag New York, Inc., 2007.

[OSM00] Osborn, Sylvia, Ravi Sandhu und Qamar Munawer: *Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies.* ACM Trans. Inf. Syst. Secur., 3(2):85–106, 2000.

[OST03] Ooi, Beng Chin, Yanfeng Shu und Kian-Lee Tan: *Relational Data Sharing in Peer-based Data Management Systems.* SIGMOD Rec., 32(3):59–64, 2003.

[OTZ+03] Ooi, Beng Chin, Kian-Lee Tan, Aoying Zhou, Chin Hong Goh, Yingguang Li, Chu Yee Liau, Bo Ling, Wee Siong Ng, Yanfeng Shu,

Xiaoyu Wang und Ming Zhang: *PeerDB: Peering into Personal Databases.* In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, Seiten 659–659. ACM Press, 2003.

[P-G] *The P-Grid Project.* `http://www.p-grid.net/`. Abgerufen am 18.06.2010.

[PAP+03] Pitoura, Evaggelia, Serge Abiteboul, Dieter Pfoser, George Samaras und Michalis Vazirgiannis: *DBGlobe: A Service-Oriented P2P System for Global Computing.* SIGMOD Rec., 32(3):77–82, 2003.

[Pie] *The Pier Project.* `http://pier.cs.berkeley.edu/`. Abgerufen am 18.06.2010.

[Plaa] *PlanetSim: Object Oriented Simulation Framework for Overlay Networks.* `http://projects-deim.urv.cat/trac/planetsim/`. Abgerufen am 18.06.2010.

[Plab] *The PlanetSim Tutorial.* `http://projects-deim.urv.cat/trac/planetsim/chrome/site/PlanetSim_tutorial.pdf`. Abgerufen am 18.06.2010.

[PS00] Park, Joon S. und Ravi Sandhu: *Binding Identities and Attributes Using Digitally Signed Certificates.* In: *ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference*, Seiten 120–127, 2000.

[R+07] Rissanen, Erik et al.: *XACML v3.0 Administrative Policy Version 1.0 Working Draft 15.* `http://www.oasis-open.org/apps/group_public/download.php/21682/xacml-3.0-administration-v1-wd-15.zip`, 2007. Abgerufen am 18.06.2010.

[Rab89] Rabin, Michael O.: *Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance.* J. ACM, 36(2):335–348, 1989.

[RB94] Reiter, Michael K. und Kenneth P. Birman: *How to Securely Replicate Services.* ACM Trans. Program. Lang. Syst., 16(3):986–1009, 1994.

[RD01] Rowstron, Antony und Peter Druschel: *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.* In: *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Seiten 329–350. Springer-Verlag, 2001.

[RF04] Rissanen, Erik und Babak Sadighi Firozabadi: *Administrative Delegation in XACML - Position Paper.* In: *W3C Workshop on Constraints and Capabilities for Web Services.* `http://www.w3.org/2004/08/ws-cc/erbsf-20040902`, 2004.

[RFH+01] Ratnasamy, Sylvia, Paul Francis, Mark Handley, Richard Karp und Scott Schenker: *A Scalable Content-Addressable Network.* In: *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, Seiten 161–172, New York, NY, USA, 2001. ACM.

[Riv91] Rivest, Ronald L.: *The MD4 Message Digest Algorithm.* In: *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 303–311, London, UK, 1991. Springer-Verlag.

[Riv92] Rivest, Ronald L.: *The MD5 Message-Digest Algorithm.* Request for Comments 1321, April 1992.

[RN05] Roth, Armin und Felix Naumann: *Benefit and Cost of Query Answering in PDMS.* In: *Revised Selected Papers International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, Seiten 50–61. Springer, August 2005.

[RN07] Roth, Armin und Felix Naumann: *System P: Completeness-driven Query Answering in Peer Data Management Systems.* In: *Proceedings Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS)*, Seiten 625–628. GI, 2007.

[RSA78] Rivest, Ronald. L., A. Shamir und L. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Commun. ACM, 21(2):120–126, 1978.

[RSRS98] Ramaswamy, Chandramouli, Ravi Sandhu, Ramouli Ramaswamy und Ravi S: *Role-Based Access Control Features in Commercial Database Management Systems.* In: *Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, Seiten 503–511, 1998.

[SBG+07] Scholl, Tobias, Bernhard Bauer, Benjamin Gufler, Richard Kuntschke, Daniel Weber, Angelika Reiser und Alfons Kemper: *HiSbase: Histogram-based P2P Main Memory Data Management.* In: *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, Seiten 1394–1397. VLDB Endowment, 2007.

[SBK+07] Scholl, Tobias, Bernhard Bauer, Richard Kuntschke, Daniel Weber, Angelika Reiser und Alfons Kemper: *HiSbase: Informationsfusion in P2P Netzwerken.* In: *Proceedings Datenbanksysteme in Business, Technologie und Web (BTW 2007), 12. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS)*, Seiten 602–605, 2007.

[SBM99] Sandhu, Ravi, Venkata Bhamidipati und Qamar Munawer: *The ARBAC97 Model for Role-Based Administration of Roles.* ACM Trans. Inf. Syst. Secur., 2(1):105–135, 1999.

[SCFY96] Sandhu, Ravi S., Edward J. Coyne, Hal L. Feinstein und Charles E. Youman: *Role-Based Access Control Models.* Computer, 29(2):38–47, 1996.

[Sch] *Webseite schuelerVZ.* `http://www.schuelervz.net/`. Abgerufen am 18.06.2010.

[Sch96] Schneier, Bruce: *Applied Cryptography.* John Wiley & Sons, 2. Auflage, 1996.

[Sch01] Schollmeier, Rüdiger: *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.* IEEE International Conference on Peer-to-Peer Computing, 0, 2001.

[SDZ08] Sturm, Christoph, Klaus R. Dittrich und Patrick Ziegler: *An Access Control Mechanism for P2P Collaborations.* In: *DaMaP '08: Proceedings of the 2008 international workshop on Data management in peer-to-peer systems*, Seiten 51–58, New York, NY, USA, 2008. ACM.

[Sha79] Shamir, Adi: *How to Share a Secret.* Commun. ACM, 22(11):612–613, 1979.

[SHS09] Sturm, Christoph, Ela Hunt und Marc H. Scholl: *Distributed Privilege Enforcement in PACS.* In: *DBSEC '09: Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference Data and Applications Security XXIII*, Seiten 142–158. Springer, Juli 2009.

[Sim91] Simon, S.: *Peer-to-Peer Network Management in an IBM SNA Network.* Network, IEEE, 5(2):30–34, Mar 1991.

[SL90] Sheth, Amit P. und James A. Larson: *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases.* ACM Comput. Surv., 22(3):183–236, 1990.

[SM98] Sandhu, Ravi und Qamar Munawer: *How to do Discretionary Access Control Using Roles.* In: *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, Seiten 47–54, New York, NY, USA, 1998. ACM.

[SM02] Sit, Emil und Robert Morris: *Security Considerations for Peer-to-Peer Distributed Hash Tables.* In: *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Seiten 261–269, London, UK, 2002. Springer-Verlag.

[SMK+01] Stoica, Ion, Robert Morris, David Karger, Frans Kaashoek und Hari Balakrishnan: *Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications.* In: *SIGCOMM '01: Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Seiten 149–160, August 2001.

[SMLN+03] Stoica, Ion, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek und Hari Balakrishnan: *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.* IEEE/ACM Trans. Netw., 11(1):17–32, 2003.

[Spi09a] *Datendiebstahl – Banken ziehen mehr als 100.000 Kreditkarten aus dem Verkehr.* `http://www.spiegel.de/wirtschaft/service/0,1518,661909,00.html`, November 2009. Abgerufen am 17.11.2009.

[Spi09b] *Leck bei britischer Tochter – Neuer Datenskandal erschüttert die Telekom.* `http://www.spiegel.de/wirtschaft/unternehmen/0,1518,661900,00.html`, November 2009. Abgerufen am 17.11.2009.

[STS08] Shudo, Kazuyuki, Yoshio Tanaka und Satoshi Sekiguchi: *Overlay Weaver: An overlay construction toolkit.* Computer Communications, 31(2):402–412, 2008. Special Issue: Foundation of Peer-to-Peer Computing.

[Stu06] Sturm, Christoph: *Orchestrating Access Control in Peer Data Management Systems.* In: *Revised Selected Papers, Current Trends in Database Technology – EDBT 2006 (Ph.D. Workshop)*, Seiten 66–74. Springer, 2006.

[STY03] Saxena, Nitesh, Gene Tsudik und Jeong Hyun Yi: *Admission control in Peer-to-Peer: design and performance evaluation.* In: *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, Seiten 104–113, New York, NY, USA, 2003. ACM Press.

[STY07] Saxena, Nitesh, Gene Tsudik und Jeong Hyun Yi: *Threshold Cryptography in P2P and MANETs: The Case of Access Control.* Comput. Netw., 51(12):3632–3649, 2007.

[Sun] *Sun's XACML Implementation.* `http://sunxacml.sourceforge.net/`. Abgerufen am 18.06.2010.

[SvHSS06] Stuckenschmidt, Heiner, Frank van Harmelen, Wolf Siberski und Steffen Staab: *Peer-to-Peer and Semantic Web.* In: Staab, Steffen und Heiner Stuckenschmidt (Herausgeber): *Semantic Web and Peer-to-Peer*, Seiten 1–17. Springer-Verlag Berlin Heidelberg, 2006.

[SZ05] Sandhu, Ravi und Xinwen Zhang: *Peer-to-Peer Access Control Architecture Using Trusted Computing Technology.* In: *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, Seiten 147–158, New York, NY, USA, 2005. ACM Press.

[Tel05] Telecommunication Standardization Sector (ITU-T): *The Directory: Authentication Framework, X.509.* Recommendation, International Telecommunication Union (ITU), August 2005.

[TEM03] Thompson, Mary R., Abdelilah Essiari und Srilekha Mudumbai: *Certificate-Based Authorization Policy in a PKI Environment.* ACM Trans. Inf. Syst. Secur., 6(4):566–588, 2003.

[TH04] Tatarinov, Igor und Alon Halevy: *Efficient Query Reformulation in Peer Data Management Systems.* In: *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, Seiten 539–550. ACM Press, 2004.

[TIM+03] Tatarinov, Igor, Zachary Ives, Jayant Madhavan, Alon Halevy, Dan Suciu, Nilesh Dalvi, Xin (Luna) Dong, Yana Kadiyska, Gerome Miklau und Peter Mork: *The Piazza Peer Data Management Project.* SIGMOD Rec., 32(3):47–52, 2003.

[Tru09] *TCG Specification Architecture Overview.* `https://www.trustedcomputinggroup.org`, November 2009. Abgerufen am 17.11.2009.

[Wal02] Wallach, Dan S.: *A Survey of Peer-to-Peer Security Issues.* In: *ISSS '02: Revised Papers, Mext-NSF-JSPS International Symposium on Software Security – Theories and Systems*, Seiten 42–57, 2002.

[WL06] Wang, Hui und Laks V. S. Lakshmanan: *Efficient Secure Query Evaluation over Encrypted XML Databases.* In: *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, Seiten 127–138. VLDB Endowment, 2006.

[WNH+06] Watanabe, Kenichi, Yoshio Nakajima, Naohiro Hayashibara, Tomoya Enokido, Makoto Takizawa und S. Misbah Deen: *Trustworthiness of Peers Based on Access Control in Peer-to-Peer Overlay Networks.* In: *26th IEEE International Conference on Distributed Computing Systems Workshops 2006*, Seite 74. IEEE Computer Society, 2006.

[WZB05] Winslett, Marianne, Charles C. Zhang und Piero A. Bonatti: *PeerAccess: a logic for distributed authorization.* In: *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, Seiten 168–179, New York, NY, USA, 2005. ACM Press.

[XL04] Xiong, Li und Ling Liu: *PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities.* IEEE Transactions on Knowledge and Data Engineering, 16(7):843–857, 2004.

[XQu] *XQuery 1.0: An XML Query Language.* `http://www.w3.org/TR/xquery/`. Abgerufen am 18.06.2010.

[YGM02] Yang, Beverly und Hector Garcia-Molina: *Improving Search in Peer-to-Peer Networks.* In: *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, Seiten 5–14, Washington, DC, USA, 2002. IEEE Computer Society.

[YGM03] Yang, Beverly und Hector Garcia-Molina: *Designing a Super-peer Network.* In: *ICDE '03: Proceedings of the 19th International Conference on Data Engineering*, Seiten 49–60, 2003.

[ZB07] Zhang, Ying und Peter Boncz: *XRPC: Interoperable and Efficient Distributed XQuery.* In: *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, Seiten 99–110. VLDB Endowment, 2007.

[Zha07] Zhang, Y.: *Towards P2P XML Database Technology.* In: *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Vienna, Austria, September 2007. PhD Workshop.

[ZLHL05] Zhang, Yu, Xianxian Li, Jinpeng Huai und Yunhao Liu: *Access Control in Peer-to-Peer Collaborative Systems.* In: *ICDCSW '05: First International Workshop on Mobility in Peer-to-Peer Systems*, Seiten 835–840, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

Anhang A

Parameter der PlanetSim Simulation

Hier werden die wichtigsten Parameter der PlanetSim Simulation vorgestellt. Neben ihrer Beschreibung werden, sofern sinnvoll, auch Standardwerte aufgeführt. Die Parameter werden hierbei unterteilt in allgemeine Simulationsparameter in Tabelle A.1, Parameter des normalen Chord Protokolls in Tabelle A.2, Parameter der Erweiterungen an Chord zum verlässlichen Chord Protokoll in Tabelle A.3 und Parameter des unstrukturierten P2P-Protokolls in Tabelle A.4. In Tabelle A.5 werden die Parameter zur Generierung der Arbeitslast einer Simulation aufgeführt.

Parameter	**Beschreibung**	**Standard-wert**
SIMULATIONSEED	Initiale Seed des Zufallszahlengenerators	
FACTORIES_NETWORKSIZE	Größe des Netzwerks beim Start der Simulation	
FACTORIES_NETWORKTOPOLOGY	Topologie des Netzwerks	RANDOM
SIMULATOR_SIMULATION_STEPS	Anzahl der Simulationsschritte, die nach einer Veränderung des Netzwerks erfolgen, bis die nächste Änderung am Netzwerk vorgenommen wird.	5
SIMULATOR_ENVIRONMENT	Schalter zum Umschalten zwischen Simulation und realer Netzwerkimplementierung. Da PACS lediglich die Simulation einsetzt, hat dieser Parameter immer den Wert SIMULATION.	SIMULATION
SIMULATOR_QUEUE_SIZE	Anzahl der Nachrichten, die ein Knoten in seiner Warteschlange für empfangene und ausgehende Nachrichten bis zu ihrer Verarbeitung puffern kann	2000
SIMULATOR_PROCESSED_MESSAGES	Anzahl der Nachrichten, die ein Peer pro Simulationsschritt verarbeiten kann	2000
BEHAVIOURS_PROPERTIES_-FAULTY_NODES	Prozentualen Anteil der bösartigen Knoten in der Simulation.	
BEHAVIOURS_PROPERTIES_-COLLABORATIVE_FAULTY_NODES	Anteil bösartig kooperierender Knoten unter den vorhandenen bösartigen Knoten im Netzwerk in Prozent	100
BEHAVIOURS_PROPERTIES_-MALICIOUS_DISTRIBUTION	Verteilung der bösartigen Knoten im Netzwerk. Da ein Peer seine ChordID nicht selbst festlegen kann, ist der Standardwert hier UNIFORM, was einer gleichmäßigen Verteilung entspricht.	UNIFORM

Tabelle A.1: Parameter der PlanetSim Simulation

Parameter	**Beschreibung**	**Standard-wert**
CHORD_STABILIZATION_STEPS	Anzahl der Simulationsschritte die vergehen, bis der nächste Stabilisierungsprozess gestartet wird	30
CHORD_FIX_FINGER_STEPS	Die Wartezeit in Simulationsschritten zwischen der Überprüfung von Fingertabelleneinträgen	30
CHORD_SUCCESSOR_LIST_SIZE	Die Größe der Successor-Liste	256
CHORD_ROUTING_SUCCESSOR_LIST_SIZE	Teil der Successor-Liste, der für das Routing der Nachrichten verwendet wird. Hierdurch wird das Verfälschen des Ergebnisses bei kleinen Netzwerken verhindert.	32
CHORD_PREDECESSOR_LIST_SIZE	Größe der Predecessor-Liste	32

Tabelle A.2: Wichtige Parameter des Chord Protokolls

Parameter	**Beschreibung**	**Standard-wert**
CHORD_MESSAGE_LIFETIME_STEPS	Anzahl an Simulationsschritten, die auf eine Antwort gewartet wird. Bei direkten Nachrichten gilt danach der Empfängerpeer als nicht mehr erreichbar. Bei Nachrichten, die geroutet werden, gilt nach Ablauf dieser Wartezeit das Routing als fehlgeschlagen. Üblicherweise wird deshalb ein neuer Versuch gestartet.	15
CHORD_FAILURE_TEST_GAMMA	γ Wert des Fehlertests	1,32
FIRSTHOPMSG	Anzahl Fingereinträge msg_hop_n, an die eine Nachricht weitergeleitet wird, für $n = 1$ (den Start des redundanten Routing Algorithmus)	12
SECONDHOPMSG	msg_hop_n für $n = 2$	12
REDUNDANTMSG	msg_hop_n für $n > 2$ beim redundantem Routing Algorithmus	1
BOOTSTRAP	Anzahl der von einem Knoten ausgewählten Bootstrap-Knoten	5
FINDNEIGHBOURMSGLIFETIME	Wartezeit für FindNeighbour-Antworten. Nach dieser Wartezeit fährt der redundante Routing Algorithmus mit Schritt vier fort (siehe Kapitel 3.5.7.3).	25
NODESETNSIZE	Größe des SKM beim redundanten Routing Algorithmus	32
MAXREPLICAS	Anzahl an Replikaten, die im Netzwerk gespeichert werden	32
MAXSAMPLES	Anzahl der Stichproben beim Stichprobentest für Chord interne Verwaltungsnachrichten	5

Tabelle A.3: Zusätzliche Parameter des verlässlichen Chord Protokolls

Parameter	**Beschreibung**	**Standard-wert**
MSGLIFETIME	Lebenszeit gerouteter Nachrichten in Simulationsschritten. Entspricht der *timeToLive* der FloodingMessage Objekte.	6
DIRECTMSGLIFETIME	Lebenszeit direkter Nachrichten in Simulationsschritten. Nach Ablauf dieser Zeit gilt der Empfänger für den Sender als nicht erreichbar.	3
FINGERSIZE	Anzahl der Fingereinträge, die jeder Peer für sich verwaltet	32
MAXHOPCOUNT	Maximale Anzahl Routingschritte für Nachrichten. Erreicht eine Nachricht diese Anzahl der Routingschritte, wird sie nicht mehr weitergeleitet.	2
MAXHOPCOUNTNODEMESSAGE	Maximale Anzahl der Routingschritte für FingerEntry-Nachrichten. Da beim Eintritt eines Knotens in das Netzwerk keine vollständige Durchdringung des Netzwerks bei der Suche nach Fingereinträgen nötig ist, kann sich diese von MAXHOPCOUNT unterscheiden.	2
SEARCHNEWNODESSTEPS	Anzahl der Simulationsschritte, die zwischen den Suchprozessen nach neuen Fingereinträgen vergehen	10
NUMBOOTSTRAP	Anzahl der Bootstrap-Knoten, die ein Knoten beim Beitritt zum Netzwerk nach weiteren Fingereinträgen anfragt	5
FIXFINGERSTEPS	Wartezeit in Simulationsschritten zwischen Überprüfungen von Fingereinträgen	2
MAXREPLICAS	Anzahl der Replikate für ein Datenobjekt. Dies bestimmt auch gleichzeitig die Größe der Replikationsgruppe.	32
RUNCHECKDELEGATES	Wartezeit in Simulationsschritten zwischen der Überprüfung von Replikationsgruppenmitgliedern	5
NUMGROUPS	Anzahl der Replikationsgruppen im Netzwerk	

Tabelle A.4: Parameter des unstrukturierten P2P-Netzwerks mit definierter Replikationsgruppe

Parameter	**Beschreibung**	**Standard-wert**
DYNAMICTIMEINTERVAL	Zeitraum in Simulationsschritten, in dem die Änderungen im Netzwerk stattfinden	500
FAILINGNODES	Anzahl der Knoten, die im DYNAMICTIMEINTERVAL-Zeitraum ausfallen, d.h. das Netzwerk ohne vorherige Ankündigung verlassen	20
LEAVINGNODES	Anzahl der Knoten, die das Netzwerk im durch DYNAMICTIMEINTERVAL festgelegten Zeitraum ordentlich verlassen	30
JOININGNODES	Anzahl der Knoten, die im DYNAMICTIMEINTERVAL-Zeitraum dem Netzwerk beitreten	50
NUMPUT	Anzahl der Einfügeanfragen an die P2P-Anwendung.	10000
NUMGET	Anzahl der Anfragen nach einem Datenobjekt an die P2P-Anwendung	10000

Tabelle A.5: Wichtige Parameter zur Generierung der Arbeitslast

Anhang B

Ergebnisse Messungen Chord

In diesem Kapitel werden die Ergebnisse der Messungen des Chord Protokolls dargestellt, die keine Berücksichtigung im Text fanden. Der Vollständigkeit halber werden sie dennoch an dieser Stelle aufgeführt.

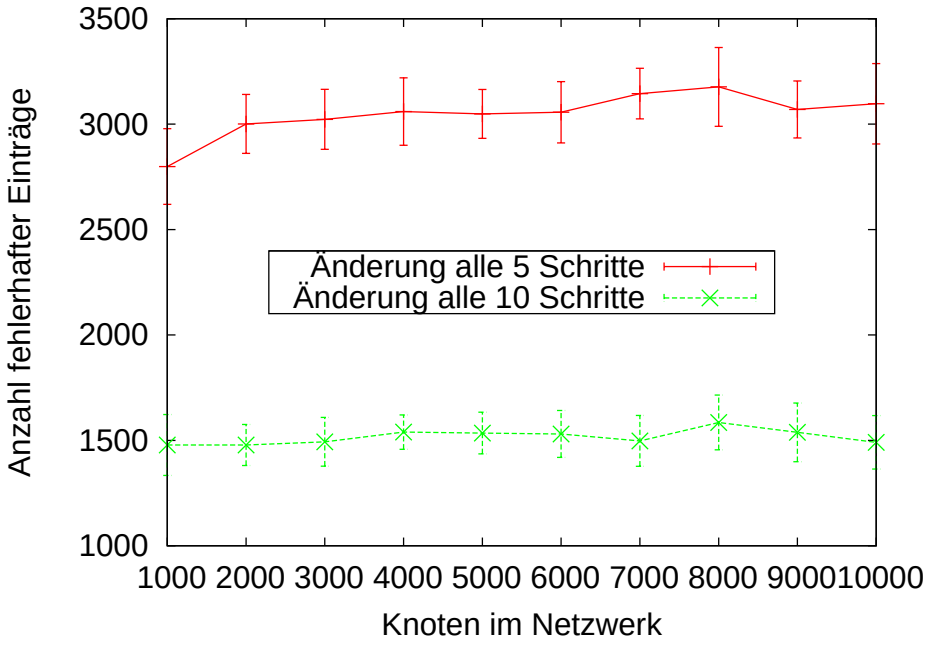


Abbildung B.1: Fehlerhafte Fingertabelleneinträge im erweiterten Chord

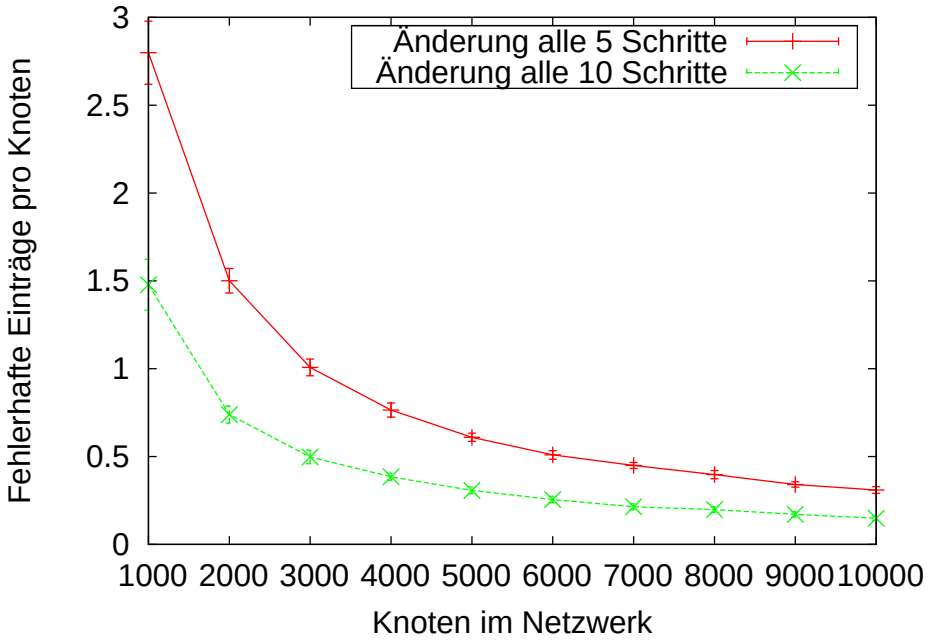
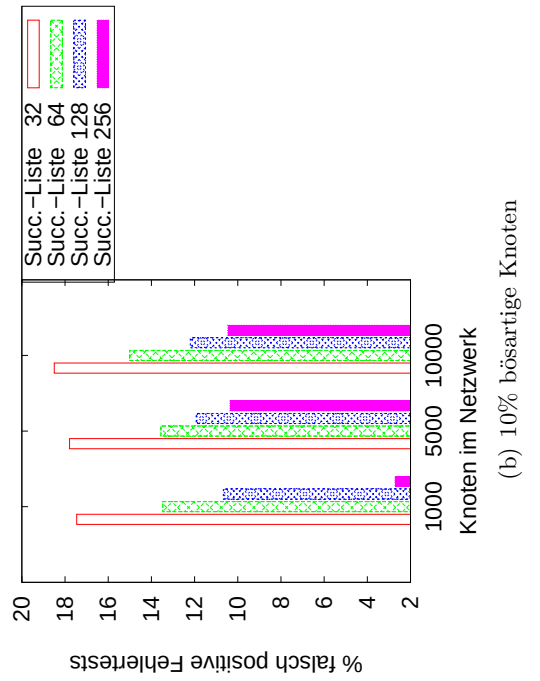


Abbildung B.2: Fehlerhafte Fingertabelleneinträge pro Knoten im erweiterten Chord

Anhang C

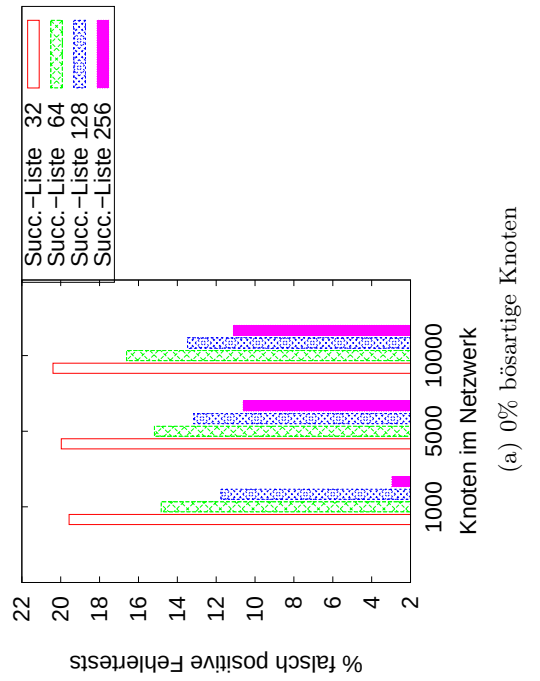
Ergebnisse Messungen Fehlertest

An dieser Stelle werden die Ergebnisse des Fehlertests, die nicht im Text behandelt werden, der Vollständigkeit halber dargestellt. Es handelt sich hierbei um die Darstellung des Fehlertests bei unterschiedlichem Anteil bösartiger Knoten in einem statischen Netzwerk unterschiedlicher Größe. In Abbildung C.1 ist der absolute Anteil falsch positiver Ergebnisse des Fehlertests für 0% (a), 10% (b), 20% (c) und 40% (d) bösartiger Peers im Netzwerk dargestellt. Der relative Anteil falsch positiver Fehlertests ist in Abbildung C.2 wiederum für 0% (a), 10% (b), 20% (c) und 40% (d) bösartiger Peers dargestellt. Die Zuverlässigkeit des Fehlertests ist schließlich in Abbildung C.3 für 0% (a), 10% (b), 20% (c) und 40% (d) bösartiger Peers dargestellt. Der Messaufbau für diesen Versuchsaufbau wird in Kapitel 5.1.2.2 vorgestellt.



(a) 0% bösartige Knoten

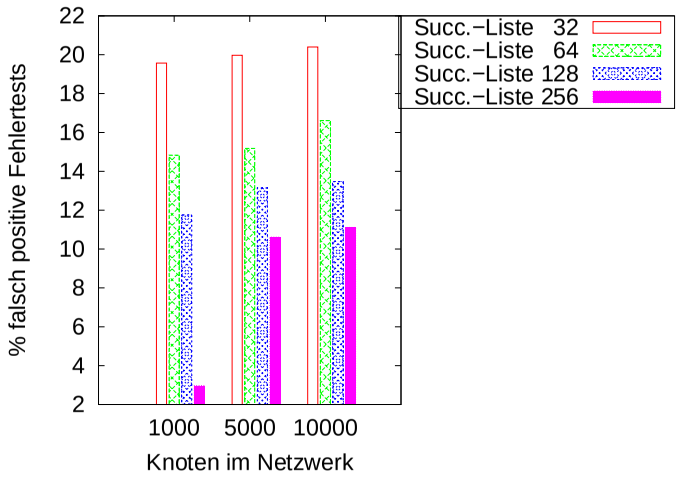
(b) 10% bösartige Knoten



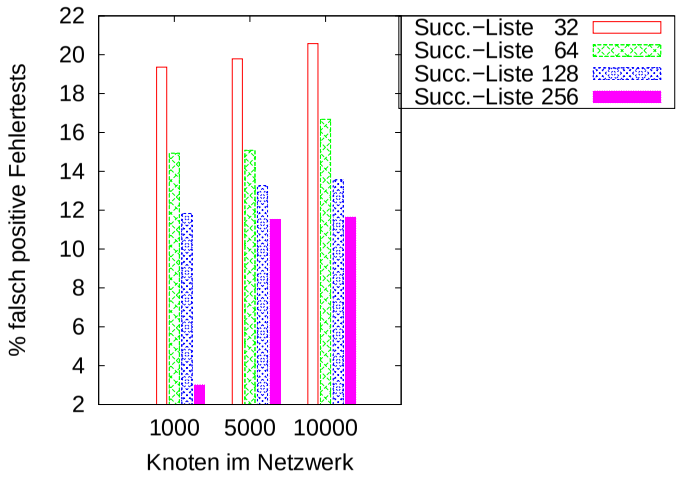
(c) 20% bösartige Knoten

(d) 40% bösartige Knoten

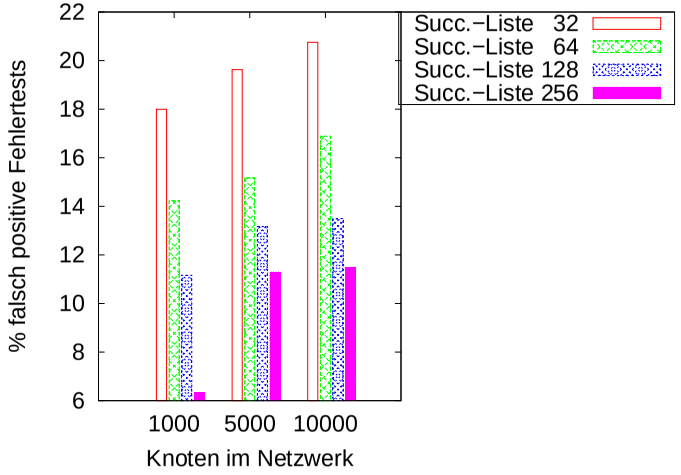
Abbildung C.1: Falsch positive Fehlertests bei unterschiedlichen Netzwerkgrößen und Anteilen bösartiger Knoten



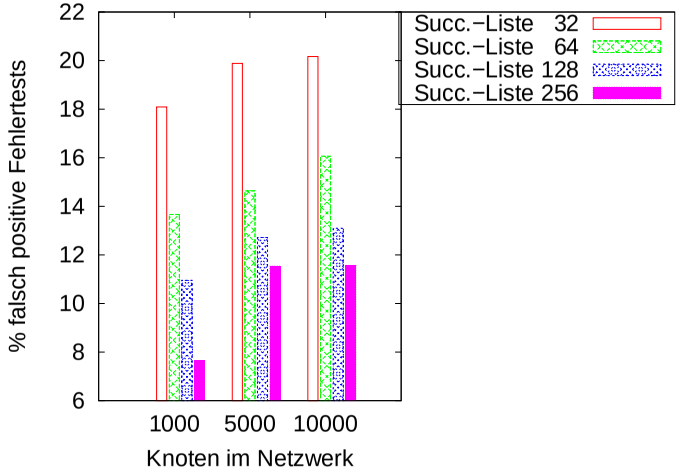
(a) 0% bösartige Knoten



(b) 10% bösartige Knoten

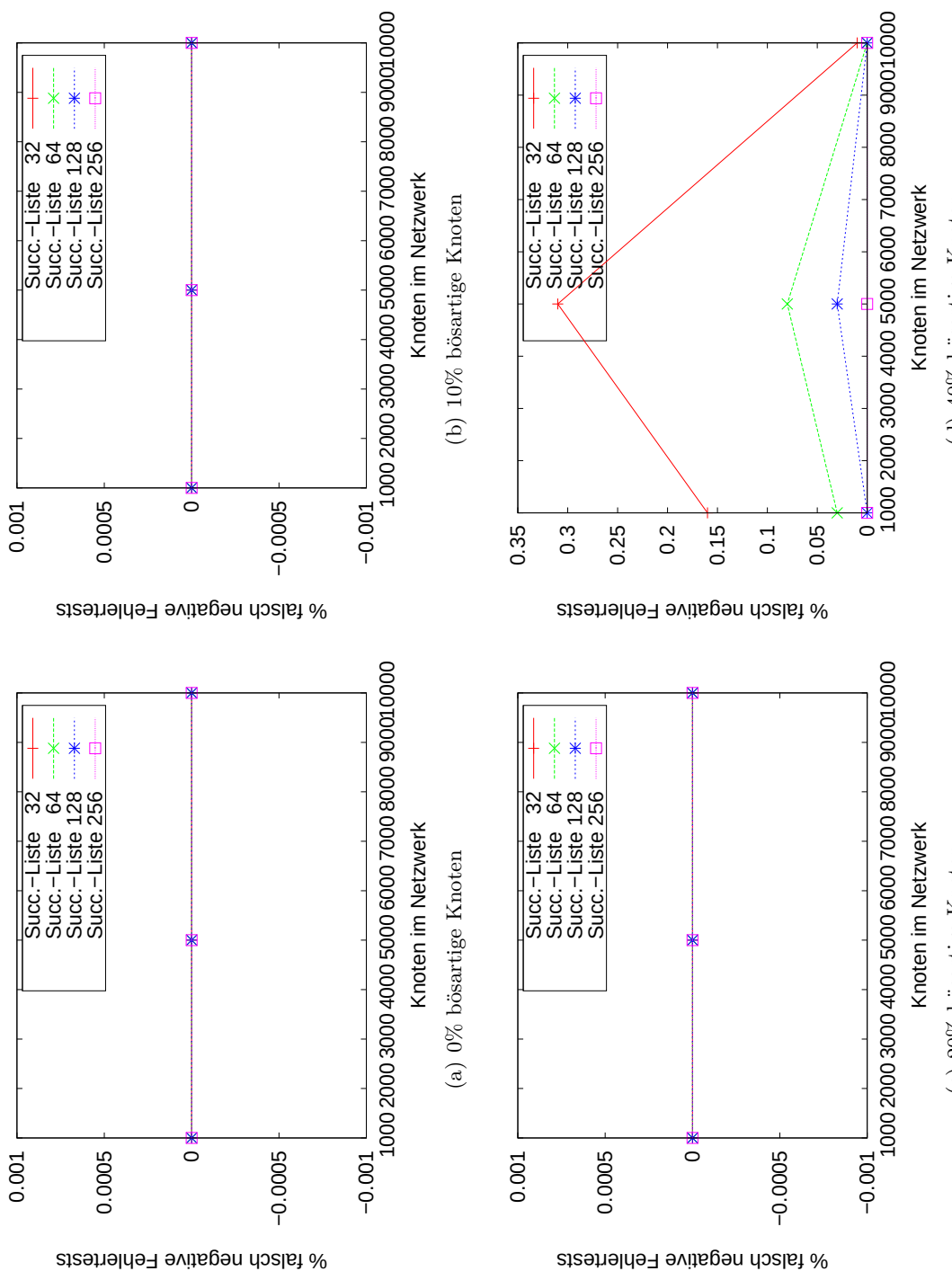


(c) 20% bösartige Knoten



(d) 40% bösartige Knoten

Abbildung C.2: Falsch positive Fehlertests bei korrekten AKMs und verschiedenen Anteilen bösartiger Knoten



(a) 0% bösartige Knoten

(b) 10% bösartige Knoten

(c) 20% bösartige Knoten

(d) 40% bösartige Knoten

Abbildung C.3: Zuverlässigkeit des Fehlertests bei verschiedenen Anteilen bösartiger Knoten und Netzwerkgrößen

Anhang D

Statistische Berechnungen zum redundanten Routing

In diesem Kapitel werden die statistischen Überlegungen kurz dargestellt, die dem redundanten Routing zugrunde liegen. Hierfür werden einige vereinfachende Annahmen getroffen. So erfordert das Routing einer Nachricht durch das Chord Protokoll immer $\frac{1}{2}\log_2(N)$ Routingschritte, wobei N der Größe des Netzwerks entspricht. Des Weiteren wird das schlechteste durch das verlässliches Chord Protokoll abgedeckte Bedrohungsszenario angenommen. Dies bedeutet, dass 30% bösartig kooperierende Peers im Netzwerk vorhanden sind. Aufgrund der zufälligen Verteilung der ChordIDs auf die Knoten kann eine gleichmäßige Verteilung dieser bösartig kooperierenden Peers angenommen werden. Deshalb ergibt sich bei 30% bösartig kooperierenden Peers im Netzwerk für jeden Peer eine Erfolgswahrscheinlichkeit von 70%, dass eine Nachricht an einen gutartigen Peer weitergeleitet wird. Wird eine Nachricht an einen bösartigen Peer weitergeleitet, ist dies gleichbedeutend mit dem Scheitern des Routings dieser Nachricht. Entsprechend ergibt sich bei jedem Routingschritt eine Erfolgswahrscheinlichkeit von 70% und eine Fehlerwahrscheinlichkeit von 30%. Die Erfolgswahrscheinlichkeit einer Anfrage beim normalen Chord basierten Routing beträgt so $0.7^{\frac{1}{2}\log_2(N)}$. Die Erfolgschancen sind deshalb schon bei kleinen Netzwerkgrößen sehr gering. Beim redundanten Routing werden die Nachrichten jeweils über verschiedene Knoten geroutet. Allerdings erfolgt das Routing der Nachrichten beim redundanten Routing am Ende zusätzlich über die Replikationsgruppe. Entsprechend erhöht sich die Anzahl der Routingschritte um eins und die Wahrscheinlichkeit a, dass eine Nachricht, die über redundantes Routing weitergeleitet wird, ihr Ziel erreicht ist $0.7^{\frac{1}{2}\log_2(N)+1}$. In Tabelle D.1 sind die Erfolgswahrscheinlichkeiten für verschiedene Netzwerkgrößen aufgelistet.

Da beim redundanten Routing die Nachricht über verschiedene unabhängige Pfade weitergeleitet wird, kann die Wahrscheinlichkeit A, dass zumindest eine Anfrage ihren Zielort erreicht, anhand einer Binomialverteilung beschrieben werden. Die Anzahl der redundanten Nachrichten *msg_hop_*1 entspricht hierbei der Anzahl der Versuche. In der Tabelle wird *msg_hop_*1 hierbei jeweils in Zehnerschritten erhöht und jeweils die Gesamtwahrscheinlichkeit A berechnet. Wie ersichtlich, sind um nahezu 100% Sicherheit zu erhalten bei einem 1000 Knoten Netzwerk 50 redundante Nachrichten nötig.

Wie aus den Ausführungen zum Chord Protokoll hervorgeht, besteht die Fingertabelle eines einzelnen Knotens aber jeweils nur aus 32 Einträgen. Es können also vom anfragenden Knoten gar keine redundanten Nachrichten an mehr als 32 verschiedene Knoten gesendet

#Knoten	Anteil Gutartig	#Routingschritte	a	msg_hop_1	A
1000	0.7	4.98	0.12	10	0.71629
1000	0.7	4.98	0.12	20	0.91951
1000	0.7	4.98	0.12	30	0.97716
1000	0.7	4.98	0.12	40	0.99352
1000	0.7	4.98	0.12	50	0.99816
5000	0.7	6.14	0.08	10	0.55721
5000	0.7	6.14	0.08	20	0.80393
5000	0.7	6.14	0.08	30	0.91318
5000	0.7	6.14	0.08	40	0.96156
5000	0.7	6.14	0.08	50	0.98298
5000	0.7	6.14	0.08	60	0.99246
5000	0.7	6.14	0.08	70	0.99666
5000	0.7	6.14	0.08	80	0.99852
10000	0.7	6.64	0.07	10	0.49185
10000	0.7	6.64	0.07	20	0.74178
10000	0.7	6.64	0.07	30	0.86878
10000	0.7	6.64	0.07	40	0.93332
10000	0.7	6.64	0.07	50	0.96612
10000	0.7	6.64	0.07	60	0.98278
10000	0.7	6.64	0.07	70	0.99125
10000	0.7	6.64	0.07	80	0.99555
10000	0.7	6.64	0.07	90	0.99774
10000	0.7	6.64	0.07	100	0.99885

Tabelle D.1: Erfolgswahrscheinlichkeit einer Anfrage bei redundantem Routing und 30% bösartiger Knoten

#Knoten	a	msg_hop_1	msg_hop_2	#Nachrichten	A
1000	0.12	12	6	50.4	0.99816
1000	0.12	12	12	100.8	1.00000
5000	0.08	12	6	50.4	0.98298
5000	0.08	12	8	67.2	0.99574
5000	0.08	10	10	70	0.99666
10000	0.07	12	12	100.8	0.99885

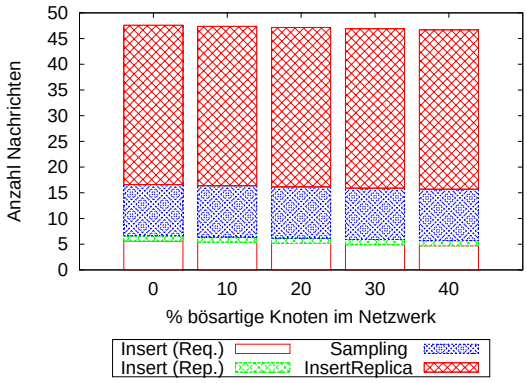
Tabelle D.2: Erfolgswahrscheinlichkeit beim mehrstufigen redundanten Routing und 30% bösartiger Knoten

werden. Die Lösung ist hierbei das vorgestellte mehrstufige Vorgehen, bei dem nicht nur am Anfang, sondern eventuell auch bei folgenden Routingschritten eine Nachricht redundant weitergeleitet wird. In dem hier betrachteten Fall reichen hierbei die ersten beiden Routingschritte aus. Allerdings muss berücksichtigt werden, dass nur Nachrichten, die an gutartige Knoten weitergeleitet werden, von diesen redundant weitergeleitet werden können. Für die in dieser Arbeit berücksichtigten Netzwerkgrößen ergibt sich deshalb das in Tabelle D.2 dargestellte Bild. Zu beachten ist, dass die Spalte *#Nachrichten* die statistisch berechnete Anzahl redundanter Nachrichten beim zweistufigen Verfahren darstellt.

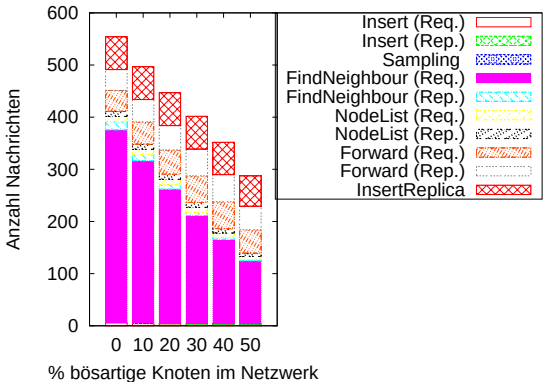
Anhang E

Messungen verlässliche DHT

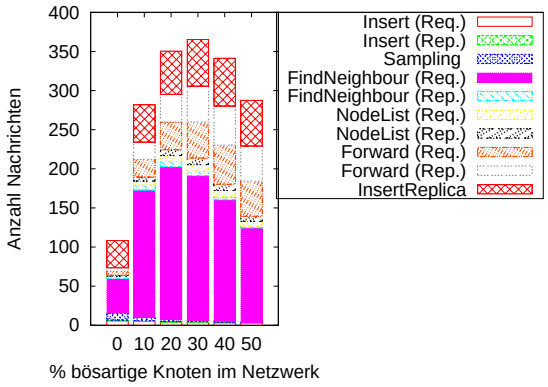
An dieser Stelle werden die verbleibenden Messungen zur verlässlichen DHT für die Netzwerkgrößen 1000 und 5000 aufgelistet. Der Simulationsaufbau für die hier dargestellten statischen Messungen entspricht jenem in Kapitel 5.1.3.1. Der Versuchsaufbau für die dynamischen Messungen wurde in Kapitel 5.1.3.2 beschrieben. Zunächst werden die statischen Messungen für Netzwerke mit 1000 (siehe Abbildung E.2) und 5000 (siehe Abbildung E.1) Knoten präsentiert. Im Anschluss sind die dynamischen Messungen für eben diese Netzwerkgrößen dargestellt (Abbildung E.4 bzw. E.3).



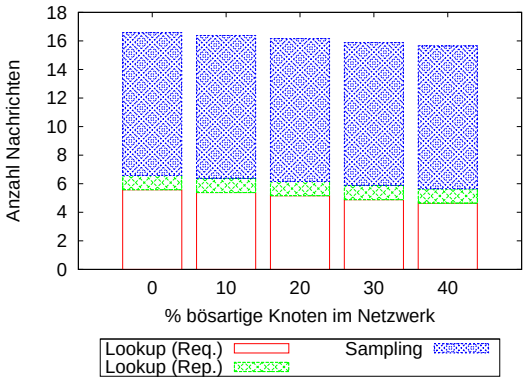
(a) Normales Routing *put*



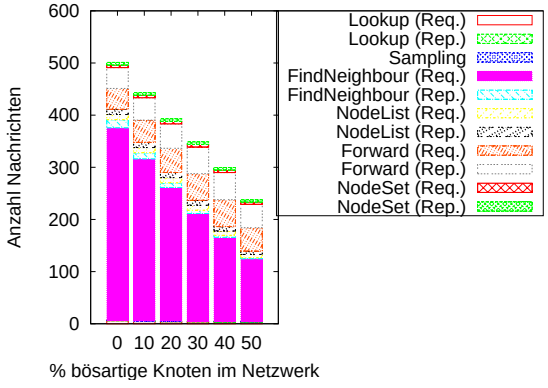
(b) Sicheres Routing *put*



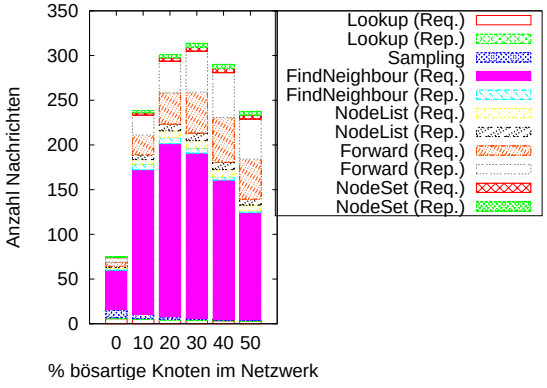
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*

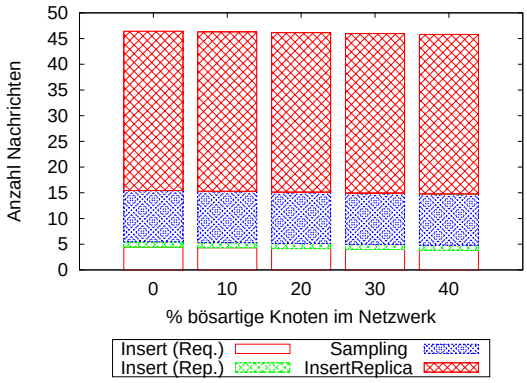


(e) Sicheres Routing *get*

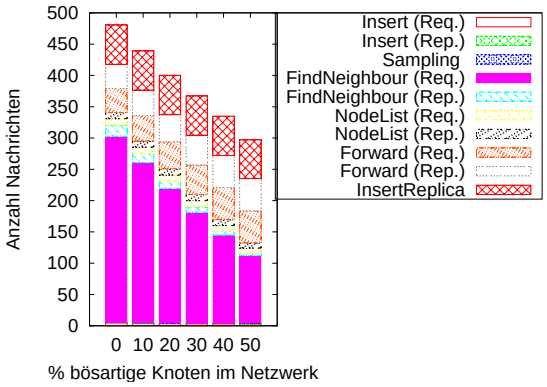


(f) Gesamtdurchschnitt *get*

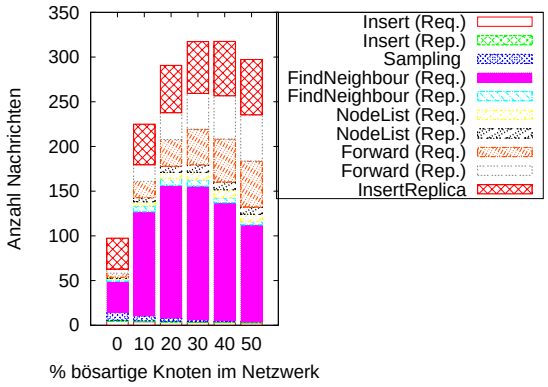
Abbildung E.1: *put* und *get* Anfragen bei 5000 Knoten im statischen Netzwerk



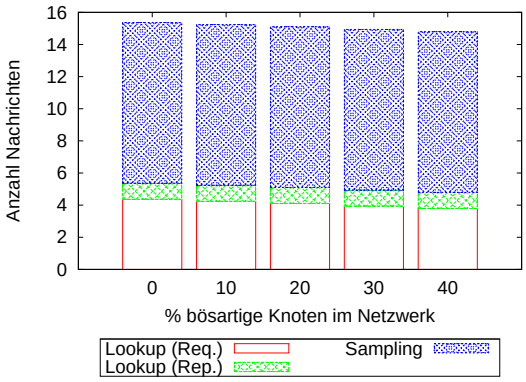
(a) Normales Routing *put*



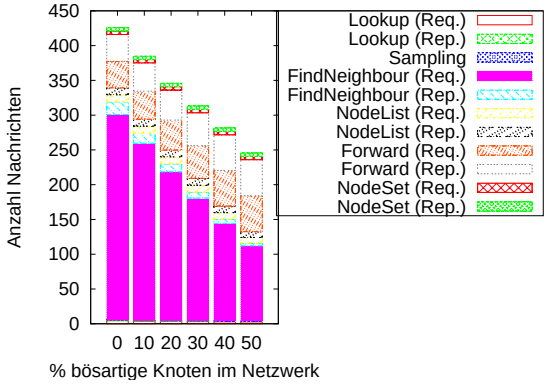
(b) Sicheres Routing *put*



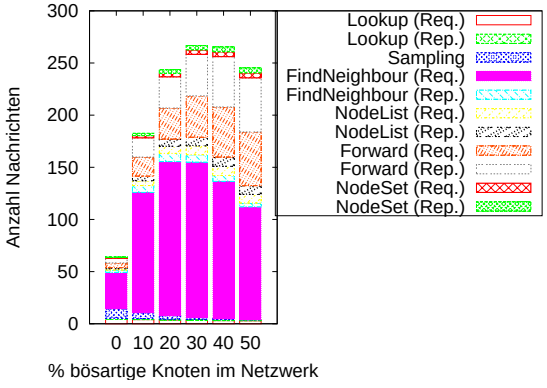
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*

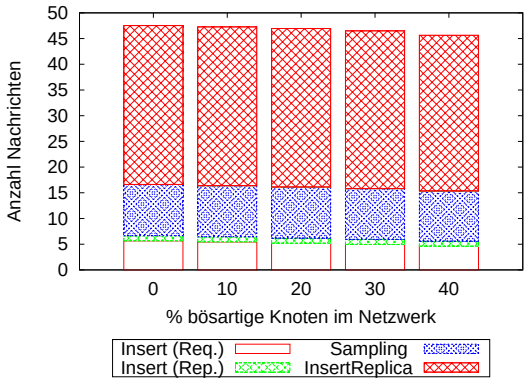


(e) Sicheres Routing *get*

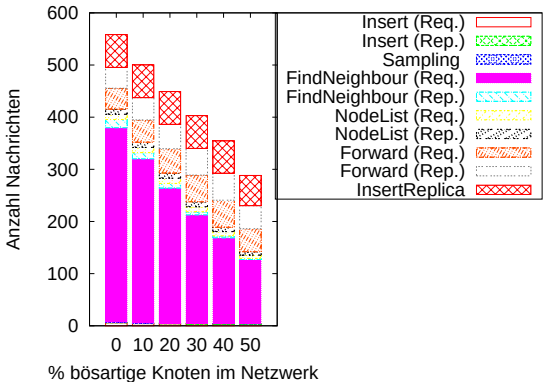


(f) Gesamtdurchschnitt *get*

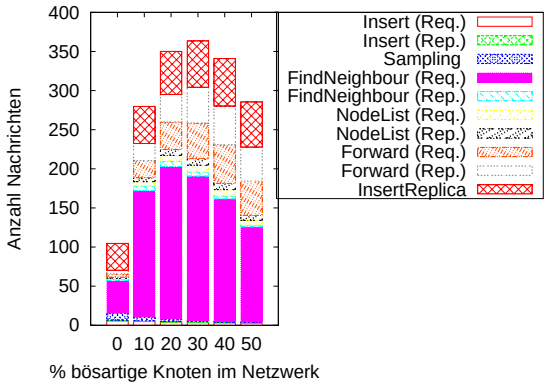
Abbildung E.2: *put* und *get* Anfragen bei 1000 Knoten im statischen Netzwerk



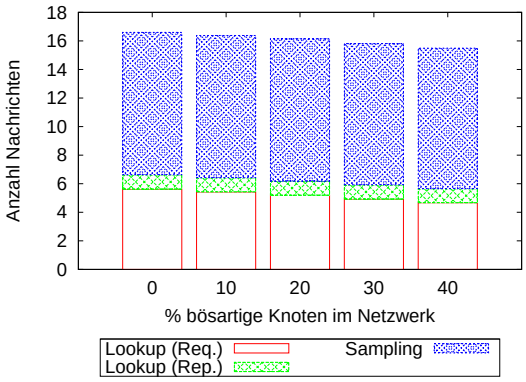
(a) Normales Routing *put*



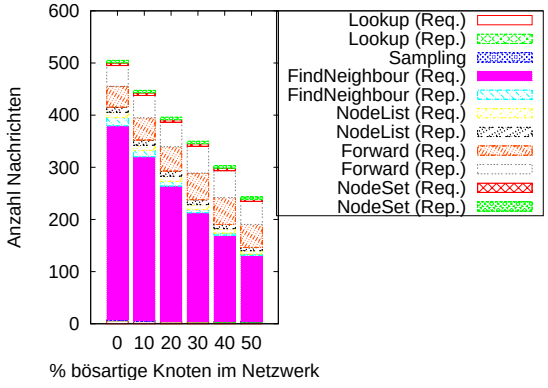
(b) Sicheres Routing *put*



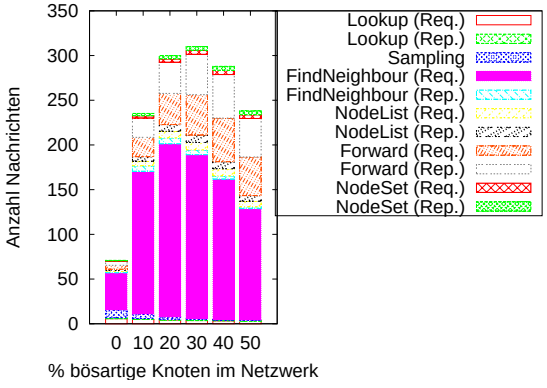
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*

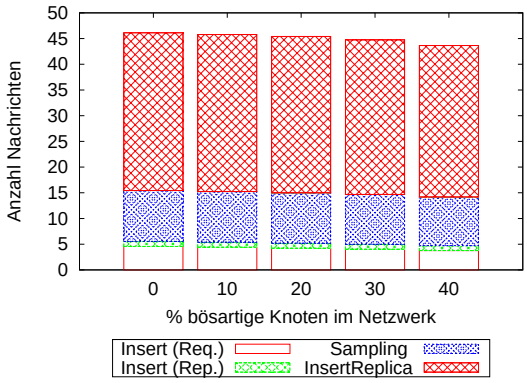


(e) Sicheres Routing *get*

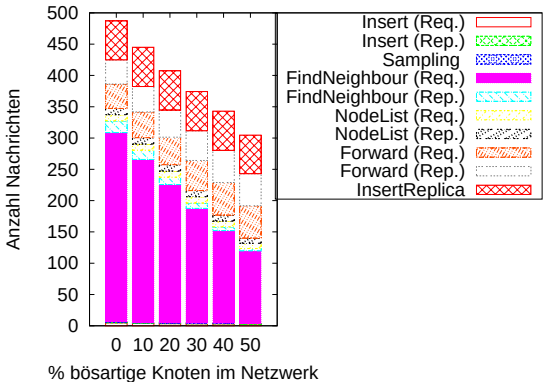


(f) Gesamtdurchschnitt *get*

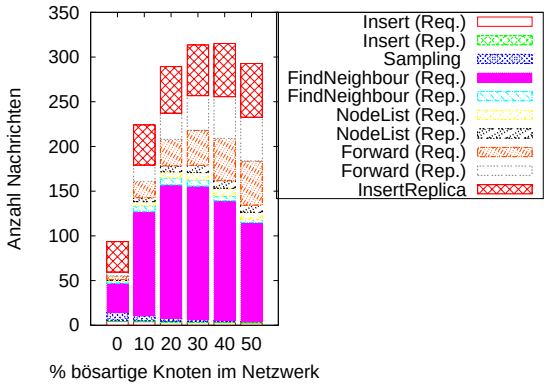
Abbildung E.3: *put* und *get* Anfragen bei 5000 Knoten im dynamischen Netzwerk



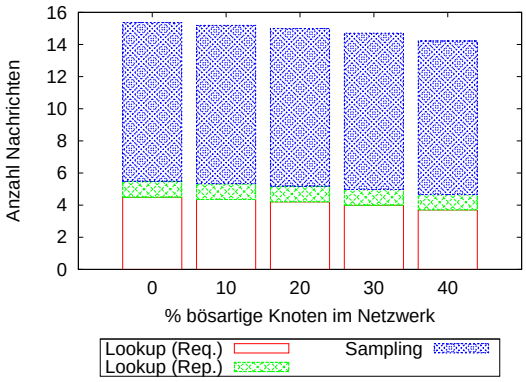
(a) Normales Routing *put*



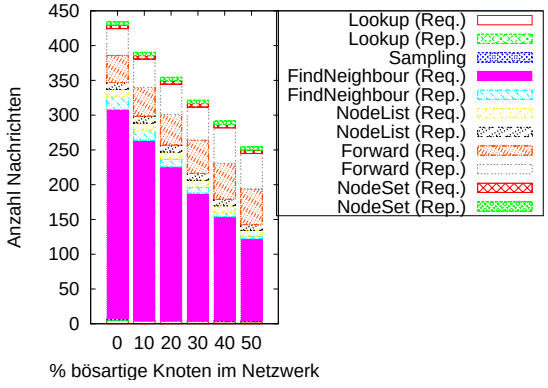
(b) Sicheres Routing *put*



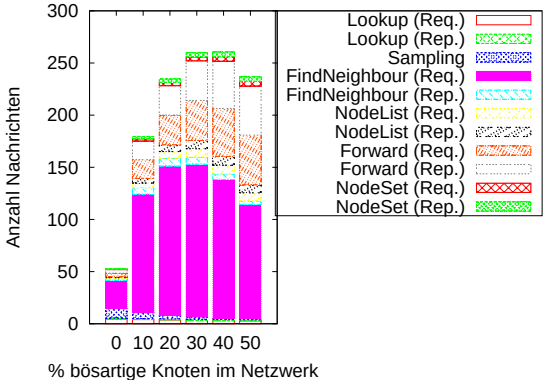
(c) Gesamtdurchschnitt *put*



(d) Normales Routing *get*



(e) Sicheres Routing *get*



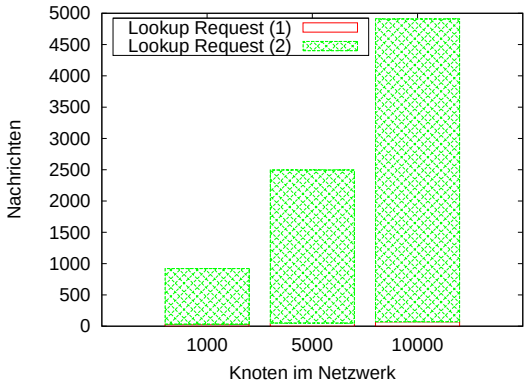
(f) Gesamtdurchschnitt *get*

Abbildung E.4: *put* und *get* Anfragen bei 1000 Knoten im dynamischen Netzwerk

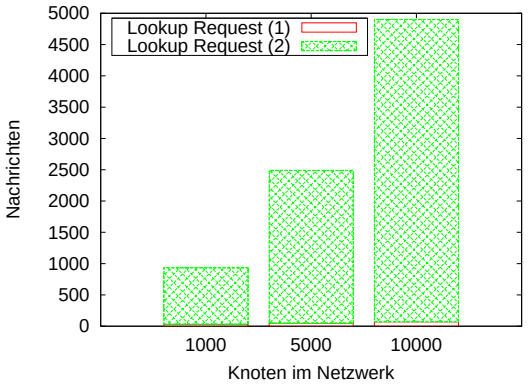
Anhang F

Messungen unstrukturiertes P2P-Netzwerk

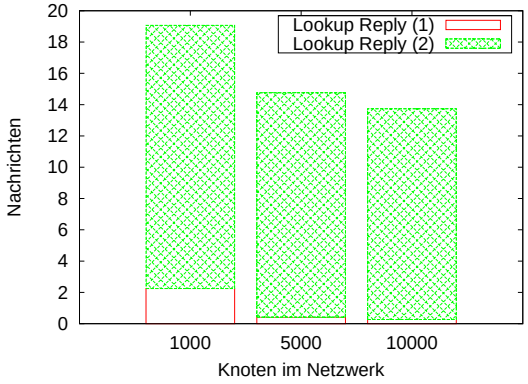
An dieser Stelle werden die verbleibenden Messungen zum unstrukturierten P2P-Netzwerk mit definierter Replikationsgruppe präsentiert. Der Messaufbau dieser Messungen wurde in Kapitel 5.2.4 vorgestellt. Hier werden die Ergebnisse für 10 bis 50% bösartiger Peers im Netzwerk für Datenanfragen und Änderungen in der Replikationsgruppe in den Abbildungen F.1,F.2 und F.3 vorgestellt. Da der Aufwand für das Speicher von Datenobjekten bei allen Anteilen bösartiger Peers im Netzwerk gleich ist wurde auf eine Darstellung verzichtet.



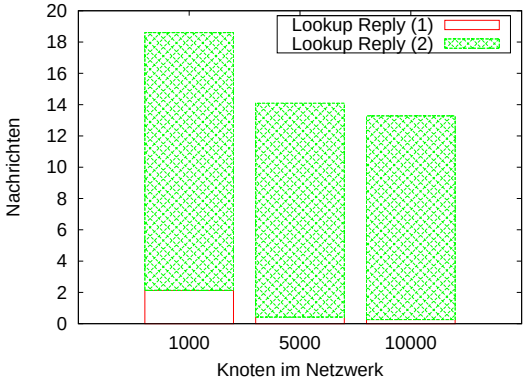
(a) Datenanfrage (Request) 10% bösartig



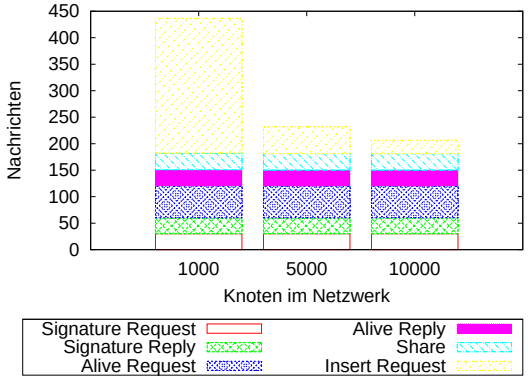
(b) Datenanfrage (Request) 20% bösartig



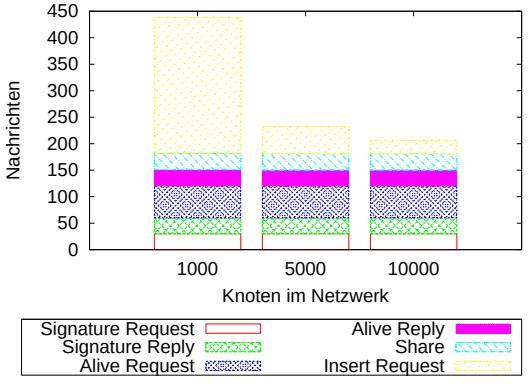
(c) Datenanfrage (Reply) 10% bösartig



(d) Datenanfrage (Reply) 20% bösartig

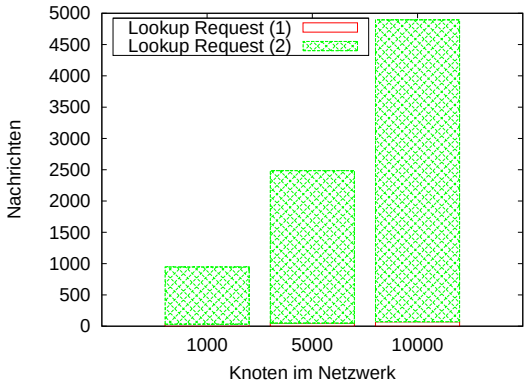


(e) Änderung Replikationsgruppe 10% bösartig

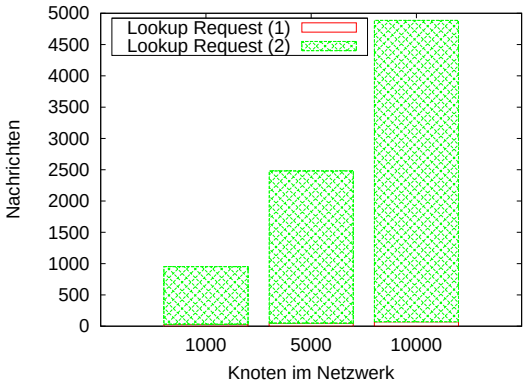


(f) Änderung Replikationsgruppe 20% bösartig

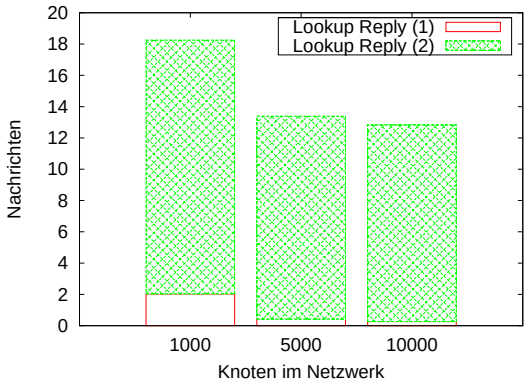
Abbildung F.1: Aufwand der Grundoperationen (10% und 20% bösartige Knoten)



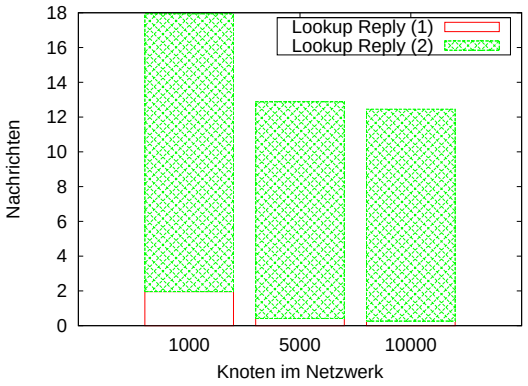
(a) Datenanfrage (Request) 30% bösartig



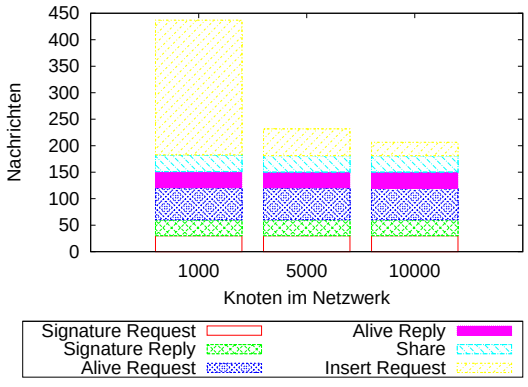
(b) Datenanfrage (Request) 40% bösartig



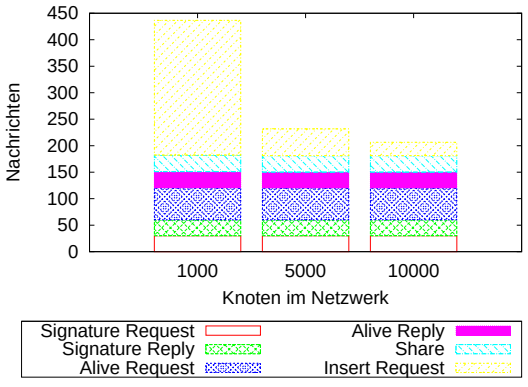
(c) Datenanfrage (Reply) 30% bösartig



(d) Datenanfrage (Reply) 40% bösartig

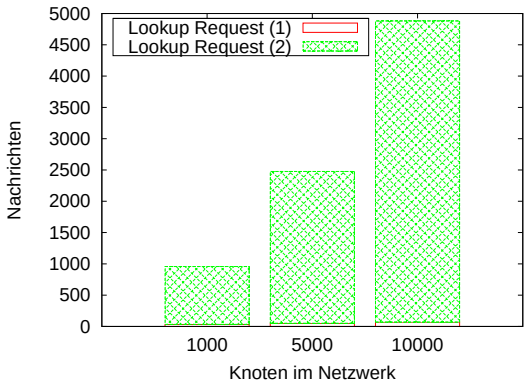


(e) Änderung Replikationsgruppe 30% bösartig

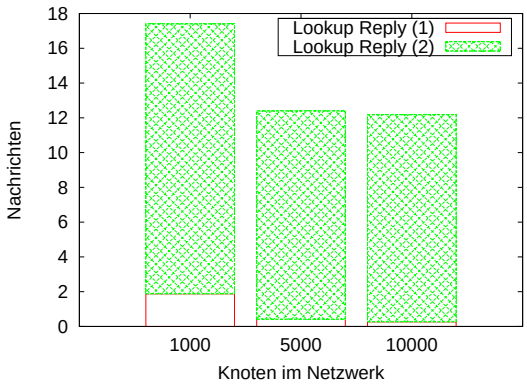


(f) Änderung Replikationsgruppe 40% bösartig

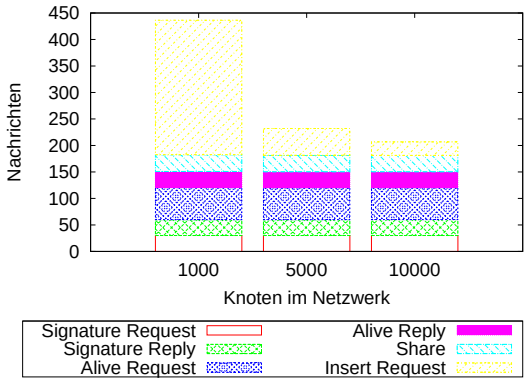
Abbildung F.2: Aufwand der Grundoperationen (30% und 40% bösartige Knoten)



(a) Datenanfrage (Request) 50% bösartig



(b) Datenanfrage (Reply) 50% bösartig



(c) Änderung Replikationsgruppe 50% bösartig

Abbildung F.3: Aufwand der Grundoperationen (50% bösartige Knoten)

Anhang G

Aufwand des Privilegienspeichers

Anhand der Simulationen konnte der Aufwand zum Speichern eines Datenobjekts in der für PACS realisierten DHT und dem für PACS realisierten unstrukturierten P2P-Netzwerk mit aktiver Replikation bestimmt werden. Darauf aufbauend kann nun der Aufwand des Privilegienspeichers bei serverseitiger und clientseitiger Durchsetzung bestimmt werden und damit der Aufwand, den PACS letztlich bei einer Verwaltung der Privilegien und bei der Durchsetzung der Privilegien bei Anfragen verursacht.

G.1 Aufwand des Privilegienspeichers bei serverseitiger Durchsetzung

Wie in Kapitel 3.7.2 dargelegt, werden bei der serverseitigen Durchsetzung lediglich Benutzerzuordnungen, administrative Privilegien von Rollen und die Export Policies im Privilegienspeicher abgelegt. Beim unstrukturierten P2P-Netzwerk kommen zusätzlich die DGIOs hinzu, die zur Verwaltung der Replikationsgruppen benötigt werden. Die sichere Speicherung der Export Policies wurde hierbei schon in Kapitel 3.5.4.5 und die von DGIOs in Kapitel 3.8.3.2 abschließend behandelt. Alle anderen Berechtigungen werden direkt beim Dateneigentümer gespeichert, bei dem auch die Datenobjekte abgelegt sind.

G.1.1 Organisation des DHT-Privilegienspeichers

Für die Organisation des DHT-Privilegienspeichers gilt:

- Benutzerzuordnungen werden unter dem Schlüssel des Rollenbezeichners in der DHT abgelegt, zu dem sie gehören.
- Administrative Privilegien von Rollen werden unter dem Rollenbezeichner in der DHT abgelegt, zu der sie gehören.
- Die Export Policies werden unter der Bezeichnung ihres Eigentümers im Netzwerk abgelegt.

Alle Benutzerzuordnungen einer Rolle und deren administrative Privilegien sind deshalb auf den gleichen Peers der DHT gespeichert. Zusammenfassend lässt sich dies wie folgt darstellen:

$$put(RollenID, Benutzerzuordnung)$$
$$put(RollenID, administrativesPrivileg)$$
$$put\_data(PeerID, ExportPolicy)$$

G.1.2 Aufbau des unstrukturierten P2P-Privilegienspeichers

Für die Organisation des unstrukturierten Privilegienspeichers gilt: Die Speicherung der Berechtigungen ist genau festgelegt. Der Speicherort wird von den Netzwerkteilnehmern selbst explizit bestimmt. Jeder Netzwerkteilnehmer legt ihn durch seine Replikationsgruppe fest. Benutzerzuordnungen werden beim Rolleneigentümer und dessen Replikationsgruppe abgespeichert. Die administrativen Privilegien von Rollen werden neben der Speicherung beim Dateneigentümer auch beim Rolleneigentümer und dessen Replikationsgruppe abgespeichert. Alle Benutzerzuordnungen einer Rolle und deren administrative Privilegien sind deshalb beim Rolleneigentümer und den Knoten seiner Replikationsgruppe gespeichert. Die Export Policy und das DGIO eines Peers werden in dessen Replikationsgruppe repliziert. Aufgrund dieser Organisation sind die Operationen des Privilegienspeichers nicht direkt auf die *get*, *put* und *delete* Operationen des unstrukturierten Netzwerks abbildbar. Die Abbildung muss deshalb erläutert werden.

G.1.2.1 Operationen des unstrukturierten P2P-Privilegienspeichers

Bei einem *get* Aufruf an den Privilegienspeicher sind für das unstrukturierte Netzwerk folgende Fälle zu unterscheiden:

Eine *get* Anfrage für eine Benutzerzuordnung impliziert zunächst eine Abfrage des DGIO für den Rolleneigentümer ($get\_data(DGIO)$). Anschließend erfolgt aufgrund der Angaben im DGIO die direkte Anfrage beim Rolleneigentümer. Sollte dieser nicht erreichbar sein, erfolgt die Anfrage bei den im DGIO angegebenen Stellvertretern.

Ist es erforderlich die Stellvertreter anzufragen, werden fünf von ihnen zufällig ausgewählt und angefragt um sicherzugehen, dass mit sehr hoher Wahrscheinlichkeit die Antwort mindestens eines gutartigen Knotens enthalten ist. Eine korrekte Antwort genügt um die anderen Antworten als inkorrekt zu erkennen (siehe Kapitel 3.8.6.4).

Das Ergebnis der Anfrage wird als Ergebnis des *get* Aufrufs an den Anfragenden zurückgegeben. Der Aufwand für einen solchen Aufruf entspricht dem Aufwand eines *get_data* Aufrufs für das DGIO plus jenem für die Nachricht an den Eigentümer des DGIO bzw. den fünf Nachrichten an die Stellvertreter.

Bei *put* Aufrufen an den Privilegienspeicher wird zunächst das DGIO mit der entsprechenden ID des Rolleneigentümers abgefragt. Ist der Besitzer des DGIO im Netzwerk erreichbar, wird die Berechtigung an diesen übermittelt. Er übernimmt die Übermittlung der Berechtigung an seine Stellvertreter. Ist der DGIO-Besitzer nicht im Netzwerk erreichbar, wird die Berechtigung direkt an alle Stellvertreter des Rolleneigentümers geschickt. Grund für die Unterscheidung ist, dass im Falle der Erreichbarkeit des Eigentümers des DGIO nur dieser die Autorisierung der Anfrage prüfen muss, während bei dessen Abwesenheit alle Stellvertreter die Autorisierung der Anfrage für sich prüfen müssen. Der Aufwand zur Speicherung entspricht in beiden Fällen dem Aufwand einer *put* Anfrage an das unstrukturierte P2P-Netzwerk. Insgesamt ergibt sich so ein Aufwand von $get\_data(DGIO) + put$.

Bei einem *delete* Aufruf ist der Ablauf deckungsgleich mit dem eines *put* Aufrufs. Anstatt des Einfügens einer neuen Berechtigung erfolgt hier jedoch das Löschen der Berechtigung. Der Aufwand hierfür ist mit dem des *put* Aufrufs identisch.

G.1.3 Überprüfung der Autorisierung

Die Autorisierung basiert auf den globalen Privilegien des Benutzers und der Export Policy des Dateneigentümers.

Handelt es sich um einen lokalen Benutzer, d.h. der Grantor ist vom selben Peer wie das Datenobjekt bzw. die Rolle der Berechtigung, die hinzugefügt werden soll, muss die Export Policy des Ursprungspeers der Rolle bzw. des Datenobjekts überprüft werden. Ansonsten erfolgt die Prüfung anhand der gespeicherten globalen Privilegien. Da die Privilegien immer beim Dateneigentümer gespeichert werden, muss diese Prüfung nicht auf den Privilegienspeicher zugreifen. Des Weiteren kann ein Benutzer aufgrund seiner Rollenzugehörigkeit für eine Aktion autorisiert sein. Die aktivierten Rollen des anfragenden Benutzers werden hierbei mit der Anfrage übermittelt. Die Überprüfung der Rollenzugehörigkeit erfolgt durch die Abfrage der Benutzerzuordnungen der angegebenen Rollen. Ist eine entsprechende Benutzerzuordnung im Privilegienspeicher vorhanden, gilt die Überprüfung der Rollenzugehörigkeit als bestanden.

Eine weitere Besonderheit stellt die Autorisierung von Widerrufen von Berechtigungen dar. Berechtigungen können nur vom Grantor widerrufen werden. Beim Entzug von Benutzerzuordnungen kann es aber zum kaskadierenden Widerruf von Berechtigungen an anderen Datenobjekten kommen. In diesem Fall gilt die entzogene Benutzerzuordnung als Autorisierung für den Widerruf. Dieser Widerruf muss durch eine Abfrage der Benutzerzuordnung oder der Export Policy des Rolleneigentümers überprüft werden. Besitzt der Benutzer, dem das Privileg entzogen werden soll, gemäß dieser Überprüfung die Rolle nicht mehr, ist der Widerruf autorisiert.

G.1.4 Aufwand des Privilegienspeichers

Bei beiden Arten des Privilegienspeichers ist die Ablage der Daten ähnlich organisiert. Alle Benutzerzuordnungen und die administrativen Privilegien einer Rolle werden auf den gleichen Peers gespeichert. Lediglich in der Bestimmung und Verwaltung dieser Gruppe von Peers und in der Speicherung der Export Policies unterscheiden sich die beiden Arten von Privilegienspeichern. Bei den folgenden Ausführungen wird soweit möglich auf die konkrete Realisierung der *put* und *get* Anfragen an den Privilegienspeicher verzichtet. Diese *put* und *get* Operationen sind entsprechend bei der Berechnung des Aufwands für den jeweiligen Privilegienspeicher durch die konkrete Realisierung zu ersetzen (siehe Kapitel G.1.2.1).

Der Aufwand für *put* und *get* Anfragen an den Privilegienspeicher ist hierbei, wie zuvor gezeigt, von der konkreten Größe des Netzwerks und dem Anteil der darin enthaltenen bösartigen Knoten abhängig.

Die Datenoperationen des Privilegienspeichers (wie z.B. *get_data*) unterscheiden sich nur dann im Aufwand von ihren entsprechenden Privilegienoperationen (*get*), falls die Verifikation der Daten aufwändiger ist, als jene der Privilegien. Dies ist weder bei den DGIOs noch bei den Export Policies der Fall (siehe Kapitel 3.8.3.2 und 3.5.4.5). Deshalb wird deren Unterscheidung für die folgenden Ausführungen aufgehoben. Bei der Berechnung des Aufwands wird jeweils der maximale und minimale Aufwand einer Operation auf dem Privilegienspeicher erläutert. Hierbei gelten folgende Abkürzungen:

Abkürzung	Bedeutung
p	Privileg
b	Benutzerzuordnung
ep	Export Policy
s	Stichprobe
d	DGIO
sig	Signatur

G.1.4.1 Erteilen neuer Berechtigungen

Beim Erteilen neuer Berechtigungen wird vor deren Speicherung die Autorisierung des Grantors überprüft. Folgende Fälle sind zu unterscheiden:

Benutzerzuordnungen Bei einer neuen Benutzerzuordnung muss der speichernde Peer überprüfen, ob der anfragende Peer ein entsprechendes administratives Privileg Benutzerzuordnung für diese Rolle besitzt. Da alle Benutzerzuordnungen für diese Rolle auf dem Peer gespeichert sind, ist hierfür keine Anfrage an den Privilegienspeicher nötig.

Handelt es sich um einen lokalen Benutzer, d.h. der Grantor ist vom selben Peer wie die Rolle, zu der ein neuer Benutzer hinzugefügt werden soll,[1] muss die Export Policy des Ursprungspeers der Rolle überprüft werden. Beim DHT-Privilegienspeicher liegt diese Export Policy nicht lokal bei den zuständigen Peers vor. Es ist deshalb eine Abfrage der Export Policy vom Privilegienspeicher notwendig. Jeder Peer, der die Benutzerzuordnung speichern soll, löst deshalb eine *get* Anfrage an den Privilegienspeicher für die Export Policy aus. Ausgehend von der Anzahl der Replikate entspricht dies 32 *get* Anfragen.

DHT	**max**	**min**
Nachrichten:	$put + 32 * get$	put
Datenmenge:	$put(p) + 32 * get(ep)$	$put(p)$

P2P	**max**	**min**
Nachrichten:	$get + put$	put
Datenmenge:	$get(d) + put(p)$	$put(p)$

Administrativen Privilegien einer Rolle Der Dateneigentümer sorgt für die Konsistenz zwischen den administrativen Privilegien von Rollen, die im Privilegienspeicher gespeichert sind, und seinen lokal gespeicherten Privilegien. Der anfragende Benutzer richtet seine Anfrage zur Gewährung eines administrativen Privilegs deshalb an den Datenobjekteigentümer. Dieser überprüft anhand der ihm vorliegenden Privilegien an den Datenobjekten, ob der anfragende Benutzer für die Erteilung autorisiert ist. Gegebenenfalls ist zur Überprüfung der Rollenzugehörigkeit eine *get* Anfrage beim Privilegienspeicher nötig (siehe Kapitel G.1.3).

Die Übertragung in den Privilegienspeicher erfolgt durch eine *put* Anfrage. Die speichernden Peers müssen lediglich sicherstellen, dass die Anfrage vom Dateneigentümer stammt.

[1] *lokaler_benutzer_rolle* ergibt wahr

DHT	**max**	**min**
Nachrichten:	$1 + get + put$	$1 + put$
Datenmenge:	$P + get(b) + put(p)$	$p + put(p)$

P2P	**max**	**min**
Nachrichten:	$1 + get + 12s + get + put$	$1 + get + put$
Datenmenge:	$p + get(b) + 6b + get(d) + put(p)$	$p + get(d) + put(p)$

Übrige Berechtigungen Alle anderen Privilegien werden direkt beim Dateneigentümer abgespeichert. Bei der Überprüfung der Autorisierung des Benutzers ist eventuell zur Überprüfung der Rollenzugehörigkeit eine *get* Anfrage an den Privilegienspeicher nötig (siehe Kapitel G.1.3).

DHT	**max**	**min**
Nachrichten:	$1 + get$	1
Datenmenge:	$p + get(p)$	p

P2P	**max**	**min**
Nachrichten:	$1 + get + 12s$	1
Datenmenge:	$p + get(b) + 6b$	p

G.1.4.2 Widerrufen von Berechtigungen

Je nach Berechtigung werden folgende drei Widerruffälle unterschieden.

Benutzerzuordnungen Mit dem Widerruf einer Benutzerzuordnung sind eventuell kaskadierende Widerrufe weiterer Benutzerzuordnungen sowie der kaskadierende Widerruf aller administrativen Privilegien der Rolle für den Benutzer verbunden, dem diese Benutzerzuordnung entzogen wird. Der kaskadierende Widerruf weiterer Benutzerzuordnungen kann lokal überprüft werden, da alle Benutzerzuordnungen einer Rolle auf denselben Peers gespeichert sind und folglich dem durchführenden Peer vorliegen.

Eine Benutzerzuordnung kann nur durch den Grantor der Benutzerbeziehung oder durch eine Änderung der Export Policy des Rolleneigentümers entzogen werden. Für die Autorisierung des anfragenden Benutzers ist deshalb nur im Fall eines DHT Privilegienspeichers die Abfrage der Export Policy des Rolleneigentümers vom Privilegienspeicher erforderlich. In allen anderen Fällen muss keine Konsultation des Privilegienspeichers durchgeführt werden.

Der kaskadierende Widerruf der administrativen Privilegien der Rolle erfolgt für jedes Privilegien einzeln nach dem festgelegten Vorgehen. Die Menge der zu entziehenden administrativen Privilegien kann der Peer wiederum lokal ermitteln.

DHT	**max**	**min**
Nachrichten:	$delete + 32 * get$	$delete$
Datenmenge:	$delete(p) + 32 * get(ep)$	$delete(p)$

P2P	**max**	**min**
Nachrichten:	$get + delete$	$get + delete$
Datenmenge:	$get(d) + delete(b)$	$get(d) + delete(b)$

Administrative Privilegien einer Rolle Alle Privilegien eines Datenobjekts sind beim Datenobjekteigentümer gespeichert. Entsprechend wird der Dateneigentümer die Autorisierung des Entzugs der Berechtigung prüfen. Hierzu ist eventuell eine *get* Anfrage an den Privilegienspeicher nötig (siehe Kapitel G.1.3).

Wurde einer Rolle ein administratives Privileg entzogen, muss dieses Privileg allen Benutzern kaskadierend entzogen werden, die diese Rolle besitzen. Hierzu ist es nötig, alle Benutzer der Rolle zu ermitteln. Diese wird durch die Abfrage der Benutzerzuordnungen aus dem Privilegienspeicher ermittelt. Alle relevanten Privilegien liegen lokal beim durchführenden Peer vor. Deshalb ist hierfür keine Zugriff auf den Privilegienspeicher nötig. Wird hierbei einer Rolle ein administratives Privileg entzogen, läuft der hier erläuterte Prozess für diese Rolle erneut ab.

Es obliegt dem Dateneigentümer das nun widerrufene administrative Privileg auch aus dem Privilegienspeicher zu löschen. Hierzu ist eine *put* Anfrage an denselben erforderlich. Der Dateneigentümer ist für eine solche Löschung automatisch autorisiert.

DHT	**max**	**min**
Nachrichten:	$1 + get + get + delete$	$1 + get + delete$
Datenmenge:	$p + get(b) + get(b) + delete(p)$	$p + get(b) + delete(p)$

P2P	**max**	**min**
Nachrichten:	$1 + get + 12s + get + 12s + delete$	$1 + get + s + delete$
Datenmenge:	$p + get(d) + 6b + get(d) + 6b + delete(p)$	$p + get(d) + b + delete(p)$

Übrige Berechtigungen Alle Privilegien eines Datenobjekts sind beim Dateneigentümer gespeichert. Zur Überprüfung der Autorisierung eines Widerrufs einer normalen Berechtigung ist eventuell eine *get* Anfrage an den Privilegienspeicher nötig. Ist der Widerruf des Privilegs autorisiert, wird der Peer den kaskadierenden Widerruf der angegebenen administrativen Privilegien durchführen. Da alle Privilegien des Datenobjekts dem Dateneigentümer lokal vorliegen, ist hierzu keine weitere Anfrage an den Privilegienspeicher nötig. Wird hierbei einer Rolle ein administratives Privileg entzogen, läuft Prozesse zum Entzug eines administrativen Privilegs einer Rolle ab.

DHT	**max**	**min**
Nachrichten:	$1 + get$	1
Datenmenge:	$p + get(b)$	p

P2P	**max**	**min**
Nachrichten:	$1 + get + 12s$	1
Datenmenge:	$p + get(d) + 6b$	p

G.1.4.3 Durchsetzung der Privilegien bei Anfragen

Bei der serverseitigen Durchsetzung ist der Dateneigentümer für die Durchsetzung der vergebenen Privilegien verantwortlich. Die Autorisierung des anfragenden Benutzers verursacht wie üblich eine eventuelle *get* Anfrage beim Privilegienspeicher. Da die Export Policy beim Dateneigentümer ebenfalls lokal vorliegt, ist für deren abschließende Auswertung auch kein

weiterer Zugriff auf den Privilegienspeicher nötig.

DHT	**max**	**min**
Nachrichten:	$1 + get$	1
Datenmenge:	$get(b)$	0

P2P	**max**	**min**
Nachrichten:	$1 + get + 12s$	1
Datenmenge:	$get(d) + 6b$	0

G.2 Aufwand des Privilegienspeichers bei clientseitiger Durchsetzung

Bei der clientseitigen Durchsetzung werden neben den Export Policies und den DGIOs alle Privilegien im Privilegienspeicher gespeichert. Die Organisation des Privilegienspeichers wurde schon in Kapitel 3.8.6.2 erläutert. Nun soll geklärt werden, welchen Aufwand diese Durchsetzungsvariante bei Verwendung des unstrukturierten P2P-Netzwerks und der DHT erzeugt.

G.2.1 Organisation des Privilegienspeichers

Bei beiden Arten des Privilegienspeichers werden alle Privilegien beim Datenobjekteigentümer und dessen Stellvertretergruppe gespeichert. Die Benutzerzuordnungen und die administrativen Privilegien von Rollen werden zudem zusätzlich beim Rolleneigentümer sowie dessen Stellvertretergruppe gespeichert.

Da die Export Policy bei der Durchsetzung der Privilegien benötigt wird, wird diese bei den Stellvertretern des Peers gespeichert. Ebenso wird das DGIO der Stellvertretergruppe bei allen Gruppenmitgliedern gespeichert.

Beim DHT-basierten Privilegienspeicher werden die DGIOs und die Export Policies zusätzlich direkt in der DHT gespeichert. Hierdurch sind sie einfacher abfragbar. Beim Privilegienspeicher des unstrukturierten Netzwerks entfällt diese zusätzliche Speicherung.

Dadurch unterscheidet sich die Durchsetzung und Verwaltung der Privilegien mit dem DHT-Privilegienspeicher bei der clientseitigen Durchsetzung nur in Details von dem unstrukturierten P2P-Privilegienspeicher mit aktiver Replikation. Insbesondere ist der Ablauf von *get*, *put* und *delete* Abfragen im Privilegienspeicher beim DHT und unstrukturierten P2P-Privilegienspeicher identisch.

Da beim DHT-Privilegienspeicher die Privilegien in den Stellvertretergruppen abgelegt werden, bezeichnen *get*, *put* und *delete* Aufrufe immer den Aufwand des unstrukturierten P2P-Netzwerks. *get_data*, *put_data* und *delete_data* hingegen bezeichnen den Aufwand eines entsprechenden DHT Aufrufs. Zu beachten ist zudem, dass der Aufwand einer *delete* bzw. *delete_data* Anfrage dem einer *put* bzw. *put_data* Operation entspricht.

G.2.2 Überprüfung der Autorisierung

Die Überprüfung der Autorisierung von Benutzern erfolgt analog zur serverseitigen Durchsetzung. Den Peers, die diese Autorisierung prüfen, liegen aufgrund der vorgestellten Organisation des Privilegienspeichers alle Privilegien zu einem Datenobjekt lokal vor. Lediglich

für die Überprüfung der Rollenzugehörigkeit eines Benutzers ist eine Anfrage an den Privilegienspeicher nötig. In diesem Fall wird wie üblich das DGIO des Rolleneigentümers aus dem Privilegienspeicher angefragt und der Rolleneigentümer bzw. bei dessen Abwesenheit fünf seiner Stellvertreter über die Rollenzugehörigkeit befragt.

Solange der Dateneigentümer bzw. Rolleneigentümer sich im Netzwerk befindet, kann er die Autorisierung stellvertretend für alle Mitglieder der Replikationsgruppe überprüfen. Die Knoten der Replikationsgruppe können sich auf sein Urteil verlassen. Ist er nicht im Netzwerk, muss jedes Replikationsgruppenmitglied die Autorisierungsprüfung für sich selbst vornehmen.

Handelt es sich um einen lokalen Benutzer, d.h. der Grantor ist vom selben Peer wie das Datenobjekt bzw. die Rolle der Berechtigung, die hinzugefügt werden soll, muss die Export Policy des Ursprungspeers des Datenobjekts bzw. der Rolle überprüft werden.

Die Autorisierung von Widerrufen von Privilegien erfolgt analog den Vorgaben der serverseitigen Durchsetzung (siehe Kapitel G.1.3).

G.2.3 Aufwand des Privilegienspeichers

Da die Organisation beim DHT-basierten Privilegienspeicher und beim unstrukturierten Privilegienspeicher nahezu identisch ist, unterscheiden sie sich lediglich im Aufwand, den eine Anfrage des Privilegienspeichers verursacht. Dieser ist abhängig von der Netzwerkgröße und dem Anteil bösartiger Knoten im Netzwerk. Die in der folgenden Betrachtung angegebenen *put* und *get* Anfragen werden entsprechend durch den für die Operation in den Simulationen gemessenen Aufwand ersetzt.

G.2.3.1 Erteilung neuer Berechtigungen

Berechtigungen werden beim Dateneigentümer und seiner Replikationsgruppe gespeichert. Diese Knoten überprüfen auch die Autorisierung bei der Erteilung neuer Privilegien.

Bei der Erteilung neuer Benutzerzuordnungen, die beim Rolleneigentümer und dessen Replikationsgruppe gespeichert werden, ist für die Überprüfung der Autorisierung des Grantors kein Zugriff auf den Privilegienspeicher nötig.

Die Erteilung eines neuen administrativen Privilegs an eine Rolle wird zunächst wie ein ganz normales Privileg behandelt. Wurde dieser Vorgang erfolgreich beendet, wird dieses zusätzlich noch an den Rolleneigentümer bzw. dessen Replikationsgruppe übertragen. Hierzu ist die Abfrage des DGIO des Rolleneigentümers erforderlich. Normalerweise ist der Dateneigentümer für diese Übertragung verantwortlich. Ist dieser nicht im Netzwerk vorhanden, erfolgt die Übertragung durch die Stellvertreter des Dateneigentümers (Mitglieder der Stellvertretergruppe). Die Stellvertreter des Rolleneigentümers müssen dann allerdings die Korrektheit dieses neuen Privilegs prüfen. Dies geschieht durch die Anforderung des DGIO des Dateneigentümers vom Privilegienspeicher. Anhand dessen kann erstens überprüft werden, ob das zu speichernde Privileg von einem Stellvertreter des Dateneigentümers kommt und zweitens, die Übertragung dieses Privilegs durch die Mehrzahl der Stellvertreter des Dateneigentümers protokolliert werden. Haben mehr als die Hälfte der Stellvertreter das neue Privileg übertragen, wird es zur Speicherung akzeptiert.

Normale Berechtigung(+p)

DHT	**max**	**min**
Nachrichten:	$get\_data + put + 32 * get\_data + 32 * 12s$	$1 + put$
Datenmenge:	$get\_data(d) + put(p) + 32 * get\_data(d) + 32 * 6b$	$p + put(p)$

P2P	**max**	**min**
Nachrichten:	$get + put + 32 * get + 32 * 12s$	$1 + put$
Datenmenge:	$get(d) + put(p) + 32 * get(d) + 32 * 6b$	$p + put(p)$

Benutzerzuordnung(+b)

DHT	**max**	**min**
Nachrichten:	$get\_data + put$	$get\_data + put$
Datenmenge:	$get\_data(d) + put(p)$	$get\_data(d)+put(p)$

P2P	**max**	**min**
Nachrichten:	$get + put$	$get + put$
Datenmenge:	$get(d) + put(p)$	$get(d) + put(p)$

Administratives Privleg einer Rolle(+apr)

DHT	**max**	**min**
Nachrichten:	[+p max. DHT] + get_data + $put + 32 * get\_data + 32 * 12s$	[+p min. DHT]+get_data+1+ put
Datenmenge:	[+p max. DHT]+$get\_data(d)$+ $put(p)$+32*$get\_data(d)$+32*$6b$	[+p min. DHT]+$get\_data(d)$+ $p + put(p)$

P2P	**max**	**min**
Nachrichten:	[+p max. P2P]+get+put+32* $get + 32 * 12s$	[+p min. P2P] + $get + 1 + put$
Datenmenge:	[+p max. P2P] + $get(d)$ + $put(p) + 32 * get(d) + 32 * 6b$	[+p min. P2P] + $get(d)$ + p + $put(p)$

G.2.3.2 Widerrufen von Berechtigungen

Der Widerruf von Privilegien kann einen kaskadierenden Widerruf weiterer Privilegien verursachen. Zur Überprüfung der Autorisierung eines Widerrufs ist möglicherweise ein Zugriff auf den Privilegienspeicher nötig.

Wird ein Privileg eines Datenobjekts entzogen, erfordert dies die erneute Verschlüsselung dieses Datenobjektes wie in Kapitel 3.8.5.2 erläutert. Für den Privilegienspeicher ergibt sich dadurch zusätzlicher Aufwand. Ist der Dateneigentümer im Netzwerk vorhanden, wird er die erneute Verschlüsselung und Aktualisierung des DGIO übernehmen. Entsprechend ist nur die Speicherung des modifizierten DGIO im Privilegienspeicher erforderlich, sowie die Datenübertragung des veränderten Datenobjekts an den Dateneigentümer. Ist der Dateneigentümer nicht im Netzwerk vorhanden, muss der Peer, der das Privileg entzieht, die erneute Verschlüsselung selbst vornehmen. Hierzu muss er von t Stellvertretern des Dateneigentümers den neuen Schlüssel anfordern. Nach der Speicherung des mit dem neuen Schlüssel verschlüsselten Datenobjekts werden die t Stellvertreter erneut informiert, um diese Verschlüsselung zu verifizieren. Für diese Verifizierung werden erneut 32 Teilsignaturen zwischen den Stellvertre-

tern ausgetauscht. Ist die Überprüfung erfolgreich, wird das DGIO der Stellvertretergruppe entsprechend aktualisiert. Hierzu ist wiederum eine Anfrage an t Stellvertreter und die Übermittlung des aktualisierten DGIO nötig. Im schlechtesten Fall müssen alle 32 Stellvertreter kontaktiert werden, um die erforderliche Anzahl korrekter Teilsignaturen zu erhalten.

Aufwand des Privilegienspeicher bei erneuter Verschlüsselung (E(do))

DHT	**max**	**min**
Nachrichten:	$2 * 32 + 32 + 32 + 2 * 32 + put\_data + put$	$put\_data + put$
Datenmenge:	$32{*}sig{+}32{*}sig{+}32{*}d{+}32{*}sig{+}put\_data(d){+}$ $put(d)$	$put\_data(d) + put(d)$

P2P	**max**	**min**
Nachrichten:	$2 * 32 + 32 + 32 + 2 * 32 + put$	put
Datenmenge:	$32 * sig + 32 * sig + 32 * d + 32 * sig + put(d)$	$put(d)$

Man unterscheidet den Widerruf von Benutzerzuordnungen, den Widerruf von administrativen Privilegien von Rollen und den Widerruf der übrigen Berechtigungen.

Benutzerzuordnungen Alle Benutzerzuordnungen sind beim Rolleneigentümer bzw. dessen Replikationsgruppe gespeichert. Zur Überprüfung der Autorisierung des Widerrufs ist keine Anfrage an den Privilegienspeicher nötig, da auch die Export Policy des Rolleneigentümers in dessen Replikationsgruppe repliziert wird.

Um den Widerruf der administrativen Privilegien der Rolle für den Benutzer zu veranlassen, müssen zunächst alle administrativen Privilegien einer Rolle ermittelt werden. Auch hierfür ist keine Abfrage des Privilegienspeichers nötig, da diese beim Rolleneigentümer (und dessen Replikationsgruppe) gespeichert werden. Der Widerruf dieser Privilegien wird dann für den betroffen Benutzer bei den jeweiligen Dateneigentümer veranlasst.

DHT	**max**	**min**
Nachrichten:	$get\_data + delete$	$get\_data + delete$
Datenmenge:	$get\_data(d) + delete(b)$	$get\_data(d) + delete(b)$

P2P	**max**	**min**
Nachrichten:	$get + delete$	$get + delete$
Datenmenge:	$get(d) + delete(b)$	$get(d) + delete(b)$

Administrative Privilegien einer Rolle Wenn einer Rolle ein administratives Privileg entzogen wird, muss dieses Privileg allen Benutzern kaskadierend entzogen werden, die diese Rolle besitzen. Die Privilegien sind beim Dateneigentümer gespeichert, der auch den Widerruf durchführt. Zur Überprüfung der Autorisierung eines Widerrufs ist eventuell eine Abfrage des Privilegienspeichers nötig.

Anschließend müssen alle Benutzer der Rolle ermittelt werden. Hierfür ist eine Abfrage der Benutzerzuordnungen für diese Rolle nötig. Dies geschieht durch die Abfrage des DGIO des Rolleneigentümers und Befragung seiner Mitglieder oder durch den Rolleneigentümer selbst.

Für alle Benutzer der Rolle wird nun ein kaskadierender Widerruf des Privilegs durchgeführt. Die hierfür relevanten Privilegien sind beim Dateneigentümer und den Mitgliedern

seiner Replikationsgruppe lokal gespeichert. Wird hierbei einer Rolle ein administratives Privileg entzogen, läuft der Prozess für diese Rolle erneut ab.

Nach dem Entzug des administrativen Privilegs muss dieses auch beim Rolleneigentümer gelöscht werden. Die Übermittlung der Änderungen erfolgt durch den Dateneigentümer oder durch seine Stellvertreter, falls dieser sich nicht im Netzwerk befindet. Das DGIO des Rolleneigentümers wurde schon zuvor vom Privilegienspeicher abgerufen. Ist der Dateneigentümer nicht im Netzwerk vorhanden, erfolgt die Übertragung durch dessen Stellvertreter. Den Stellvertretern des Rolleneigentümers obliegt es dann, die Korrektheit der Löschung des Privilegs zu prüfen. Hierzu fordern sie das DGIO des Rolleneigentümers vom Privilegienspeicher an. Mit diesem kann geprüft werden, ob das zu speichernde Privileg von einem Stellvertreter des Dateneigentümers kommt. Zudem wird die Übertragung dieses Privilegs durch die Mehrzahl der Stellvertreter des Dateneigentümers protokolliert. Haben mehr als die Hälfte der Stellvertreter das neue Privileg übertragen, wird es zur Speicherung akzeptiert.

DHT	**max**	**min**
Nachrichten:	$get\_data + delete + 32 * get\_data + 32 * 12s + 32 * get\_data + 32 * delete + 32 * get\_data + E(do)$	$1 + delete + get\_data + 1 + delete + E(do)$
Datenmenge:	$get\_data(d) + delete(p) + 32 * get\_data(d) + 32 * 6b + 32 * get\_data(d) + 32 * delete(p) + 32 * get\_data(d) + E(do)$	$p + delete(p) + get\_data(d) + p + delete(p) + E(do)$

P2P	**max**	**min**
Nachrichten:	$get\_data + delete + 32 * get\_data + 32 * 12s + 32 * get\_data + 32 * delete + 32 * get\_data + E(do)$	$1 + delete + get\_data + 1 + delete + E(do)$
Datenmenge:	$get\_data(d) + delete(p) + 32 * get\_data(d) + 32 * 6b + 32 * get\_data(d) + 32 * delete(p) + 32 * get\_data(d) + E(do)$	$p + delete(p) + get\_data(d) + p + delete(p) + E(do)$

Übrige Berechtigungen Vor dem Widerruf einer Berechtigung muss zunächst das DGIO des Dateneigentümers abgefragt werden. Ist der Dateneigentümer nicht im Netzwerk, muss der Aufruf zum Widerruf an alle Stellvertreter gesendet werden. Zur Autorisierung eines Widerrufs ist gegebenenfalls eine Anfrage an den Privilegienspeicher nötig. Ist der Widerruf der Berechtigung autorisiert, wird der Peer den kaskadierenden Widerruf der Berechtigung durchführen. Alle nötigen Privilegien liegen dem Peer hierzu lokal vor. Wird hierbei einer Rolle ein administratives Privileg entzogen, läuft der Prozess zum Entzug eines administrativen Privilegs einer Rolle ab.

DHT	**max**	**min**
Nachrichten:	$get\_data + delete + 32 * get\_data + 32 * 12s + E(do)$	$1 + delete + E(do)$
Datenmenge:	$get\_data(d) + delete(p) + 32 * get\_data(d) + 32 * 6b + E(do)$	$p + delete(p) + E(do)$

P2P	**max**	**min**
Nachrichten:	$get\_data + delete + 32 * get\_data + 32 * 12s + E(do)$	$1 + delete + E(do)$
Datenmenge:	$get\_data(d) + delete(p) + 32 * get\_data(d) + 32 * 6b + E(do)$	$p + delete(p) + E(do)$

G.2.3.3 Durchsetzung der Privilegien bei Anfragen

Die clientseitige Durchsetzung ist in Kapitel 3.8.4 für Datenabfragen, in Kapitel 3.8.5.1 für das Einfügen neuer Datenobjekte und in Kapitel 3.8.5.3 für das Verändern bestehender Objekte genauer erläutert.

Für den Privilegienspeicher geht es hier nur um die Übermittlung der Schlüssel und Verwaltung der Export Policies und DGIOs. Alle Stellvertreter besitzen das DGIO ihrer Replikationsgruppe, alle Privilegien der Datenobjekte des Dateneigentümers, zu dem sie gehören, sowie dessen Export Policy. Aufgrund des Einsatzes der Schwellensignatur müssen mindestens t Stellvertreter kontaktiert werden um einen Schlüssel zu erhalten. Jeder dieser t Stellvertreter prüft die Autorisierung des anfragenden Benutzers für sich alleine. Lediglich zur Überprüfung der Rollenzugehörigkeit eines Benutzers ist hierbei eine Anfrage an den Privilegienspeicher notwendig.

Datenabfragen Alle Datenobjekte sind verschlüsselt gespeichert. Der anfragende Benutzer muss zunächst das DGIO abfragen. Dann kann er den Schlüssel von den Stellvertretern des Dateneigentümers anfordern. Jeder angefragte Stellvertreter wird die Autorisierung des anfragenden Benutzers prüfen. Hierbei werden wie erläutert eventuell *get* Anfragen an den Privilegienspeicher benötigt.

DHT	**max**	**min**
Nachrichten:	$get\_data + 2 * 32 + 32 * get\_data + 32 * 12s$	1
Datenmenge:	$get\_data(d) + 32 * sig + 32 * get\_data(d) + 32 * 6b$	sig

P2P	**max**	**min**
Nachrichten:	$get + 2 * 32 + 32 * get + 32 * 12s$	1
Datenmenge:	$get(d) + 32 * sig + 32 * get(d) + 32 * 6b$	sig

Einfügen neuer Daten Das Einfügen neuer Daten kann nur vom Dateneigentümer vorgenommen werden. Damit der Zugriff auf das neue Datenobjekt eingeschränkt werden kann, muss das neue Datenobjekt in die Export Policy und das DGIO des Dateneigentümers aufgenommen werden. Entsprechend ist die Speicherung der veränderten Export Policy und des DGIO im Privilegienspeicher notwendig.

DHT	**max**	**min**
Nachrichten:	$put\_data + put + put\_data + put$	$put\_data + put + put\_data + put$
Datenmenge:	$get\_data(d) + put(d) + put\_data(ep) + put(ep)$	$put\_data(d) + put(d) + put\_data(ep) + put(ep)$

P2P	**max**	**min**
Nachrichten:	$put + put$	$put + put$
Datenmenge:	$put(d) + put(ep)$	$put(d) + put(ep)$

Veränderung eines Datenobjekts Beim Modifizieren eines Datenobjekts fällt zunächst der Aufwand zur Datenabfrage des Datenobjekts an. Da das Datenobjekt do_p verändert wurde, ändert sich dessen selbst verifizierender Schlüssel $ID^{SV}_{do_p}$. Deshalb muss das DGIO des Datenobjekteigentümers verändert werden. Ist dieser im Netzwerk, kann er die Autorisierung für das Verändern des Datenobjekts und die Anpassung des DGIO selbst vornehmen. Ansonsten müssen Stellvertreter die Modifikation des DGIO bestätigen. Auch hierfür sind wiederum Bestätigungen von t aus 32 Stellvertretern nötig. Jeder der Stellvertreter prüft, ob der anfragende Benutzer für eine Modifikation des Datenobjekts autorisiert ist. Hierbei fallen die für eine solche Prüfung eventuell nötigen Zugriffe auf den Privilegienspeicher an. Erhält der anfragende Peer die nötigen Teilsignaturen, kann er das nun unterschriebene DGIO den Stellvertretern zur Speicherung übermitteln.

DHT	**max**	**min**
Nachrichten:	[Datenabfrage DHT max.] $+ 2 * 32 + 32 * get\_data + 32 * 12s + put\_data + put$	[Datenabfrage DHT min.] $+ 1 + put\_data + put$
Datenmenge:	[Datenabfrage DHT max.] $+ 32d + 32 * sig + 32 * get\_data(d) + 32 * 6b + put\_data(d) + put(d)$	[Datenabfrage DHT min.] $+ put\_data(d) + put(d)$

P2P	**max**	**min**
Nachrichten:	[Datenabfrage P2P max.] $+ 2 * 32 + 32 * get + 32 * 12s + put$	[Datenabfrage P2P min.] $+ 1 + put$
Datenmenge:	[Datenabfrage P2P max.] $+ 32d + 32 * sig + 32 * get\_data(d) + 32 * 6b + put(d)$	[Datenabfrage P2P min.] $+ put(d)$

Lebenslauf

Nachname:	Sturm
Vorname:	Christoph
Geburtstag:	18.09.1976
Geburtsort:	Bad Mergentheim, Deutschland

1983 - 1987	Grundschule Bad Mergentheim
1987 - 1993	Kopernikus Realschule Bad Mergentheim
1993 - 1996	Wirtschaftsgymnasium Bad Mergentheim
1996 - 1997	Wehrdienst
1997 - 2000	Berufsakademie Mosbach
	Abschluss als Diplom Wirtschaftsinformatiker (BA)
2000 - 2003	Universität Konstanz
	Abschluss als Master of Science in Information Engineering
2004 - 2010	Universität Zürich
	Wissenschaftlicher Mitarbeiter und Doktorand am Institut für Informatik
2010	Dissertation „Dezentral koordinierte Zugriffskontrolle in Peer-to-Peer Datenbanken“